XAPP1030 (v1.0.1) May 6, 2008

Reference System: PLBv46 Endpoint Bridge for PCI Express in a ML505 Embedded Development Platform

Author: Lester Sanders

Abstract

This reference system demonstrates the functionality of the PLBv46 Endpoint Bridge for PCI Express® used in the Xilinx ML505 Embedded Development Platform. The PLBv46 Endpoint Bridge uses the Xilinx Endpoint core for PCI Express in the Virtex®-5 XC5VLX50T FPGA.The PLBv46 Bus is an IBM CoreConnect bus used for connecting IBM PPC405 and PPC440 and the MicroBlaze microprocessors to Xilinx IP cores.

A variety of tests generate and analyze PCIe® traffic for hardware validation of the PLBv46 Endpoint Bridge. PCIe transactions are generated and analyzed by Catalyst and LeCroy test equipment. For endpoint to root complex transactions, the pcie_dma software application generates DMA transactions which move data over the PCIe link. For root complex to endpoint transactions, Catalyst and LeCroy scripts generate PCIe traffic. A Catalyst script which configures the PLBv46 Endpoint Bridge and performs memory write/read transactions is discussed. The steps to use Catalyst to measure PCIe performance are given, and performance results are provided. The principal function of the section on performance measurements is to show how performance measurements are made.

Two stand-alone tools, PCItree and Memory Endpoint Test, are used to write and read PLBv46 Endpoint Bridge configuration space and memory in a PC environment. This is the least expensive and easiest way to learn about a PCIe hardware test environment.

The use of the ChipScope™ tool in debugging PLBv46 Endpoint Bridge issues is described.

Included System

The reference system for the PLBv46 Endpoint Bridge in the ML505 Embedded Development Platform is available at:

http://www.xilinx.com/support/documentation/application_notes/xapp1030.zip

The reference system in the xapp1030.zip file is described on page 2.

Introduction

The PLBv46 Endpoint Bridge is a PCIe endpoint instantiated in a Xilinx FPGA which communicates with a root complex. The reference systems are tested using commercial test equipment from LeCroy and Catalyst. LeCroy and Catalysts are two Analyzers/Exercisers used to verify PCIe systems. The Catalyst and LeCroy testers allow generation, analysis, capture, and triggering of Translation Layer, Data Link Layer, and Physical Layer packets. The reference systems are also tested in two test environments which are inexpensive and PC based.

The PLBv46 Endpoint Bridge is tested using the LeCroy and Catalyst testers as root complex. The ML505 Evaluation Board is inserted into the LeCroy or Catalyst PCIe slots for testing. Sample Catalyst scripts are provided in the `ml505_mb_plbv46_pcie/catalyst` directory. Sample Lecroy scripts are provided in the `ml505_mb_plbv46_pcie/lecroy` directory.

The tests for the PLBv46 Endpoint Bridge which do not require LeCroy or Catalyst test equipment are the PCIE Configuration Verification (PCIE CV), PCItree, and the Memory EndPoint Test (MET) tests. These are run using the ml505_mb_plbv46_pcie project. These tests are quick to setup and costs nothing other than a PC with PCIe slots. For these tests, the

© 2008 Xilinx, Inc. All rights reserved. XILINX, the Xilinx logo, and other designated brands included herein are trademarks of Xilinx, Inc. All other trademarks are the property of their respective owners.

ml505 Embedded Development Platform is inserted into the x1 PCIe slot of a PC (Dell 390). The PC based PCItree and/or MET software are installed. The PCItree Bus Viewer (www.pcitree.de) and the Xilinx MET tests allow the user to write and read ML505 memory with any pattern, with different lengths. PCItree and the MET do not provide the capability to analyze PCIe traffic.

Hardware and Software Requirements

The hardware and software requirements for this reference system are:

- Xilinx ML505 Rev A board
- Xilinx Platform USB or Parallel IV programming cable
- Serial communication cable and serial communication utility (TeraTerm)
- Xilinx Platform Studio 10.1i
- Xilinx Integrated Software Environment (ISE®) 10.1
- Xilinx ChipScope Pro 10.1i
- Catalyst SPX Series PCI Express Bus Protocol Analyzer/Exerciser
- LeCroy PETracer Analyzer / PETrainer Exerciser

Reference System Specifics

This reference system includes the MicroBlaze™ Processor, MPMC, XPS BRAM, XPS INTC, XPS GPIO, XPS UART Lite, XPS Central DMA, and PLBv46 Endpoint Bridge. Both the processor and the bus run at a frequency of 125 MHz. The MicroBlaze processor uses 2 KB for the instruction cache (I-cache) and 4 KB for the data cache (D-cache). MPMC runs at a frequency of 125 MHz and is set up for three ports.

Figure 1 is the block diagram of the reference system.

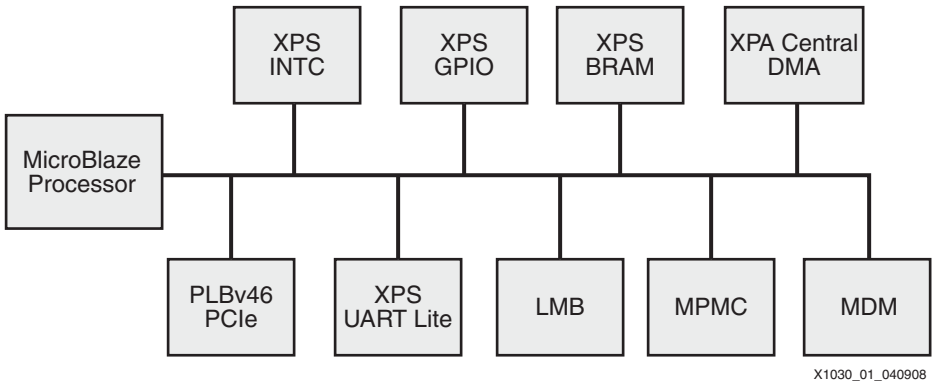


Figure 1: **Block Diagram of Reference System**

Table 1 provides the address map of the system.

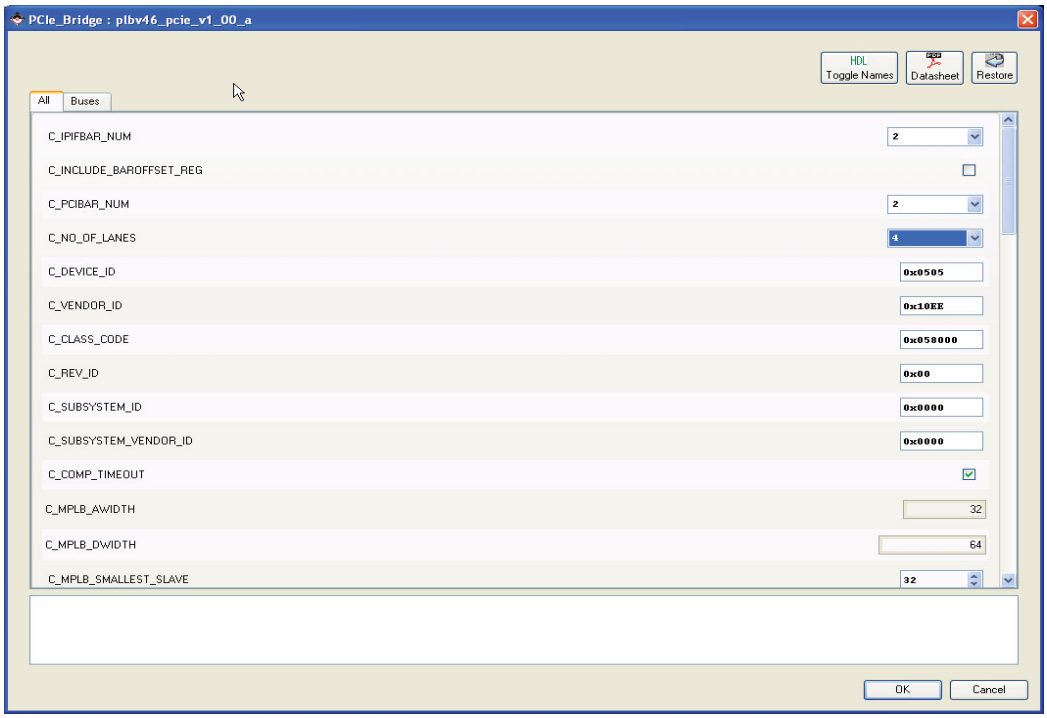
Table 1: **Reference System Address Map**

Peripheral	Instance	Base Address	High Address
MDM	debug_module	0x84400000	0x8440FFFF
XPS INTC	xps_intc_0	0x81800000	0x8180FFFF
XPS GPIO	xps_gpio_0	0x81400000	0x8140FFFF
XPS BRAM CNTLR	xps_bram_if_cntlr_1	0x8AE10000	0x8AE1FFFF
XPS Central DMA	xps_cdma_0	0x80200000	0x8020FFFF
PLBv46 Endpoint Bridge	plbv46_pcie_0	0x85C00000	0x85C0FFFF

Table 1: **Reference System Address Map**

Peripheral	Instance	Base Address	High Address
XPS Uartlite	RS232	0x84000000	0x8400FFFF
LMB Cntlr	ilmb_cntlr	0x00000000	0x00001FFF
LMB Cntlr	dlmb_cntlr	0x00000000	0x00001FFF
MPMC	DDR2_SDRAM_32Mx32	0x90000000	0x9FFFFFFF

In XPS, double click on PCIe_Bridge in the System Assembly View to invoke the PLBv46 _PCIe generics editor. The generics shown in Figure 2 are used to configure the PLBv46 Endpoint Bridge. The Xilinx Device ID = 0x0505 and Vendor ID = 0x10EE are displayed in many of the PCIe tests done in this application note.



X1030_02_040908

Figure 2: **PLBv46 Endpoint Bridge Parameters**

Implementation Results

The resource utilization in the reference design is shown in Table 2.

Table 2: **Design Resource Utilization**

Resources	Used	Available	Utilization (%)
Slice Registers	11984	28800	41
Slice LUTs	12247	28800	42
DCM_ADV	2	12	12
Block RAM	56	60	93

ML505 Setup

Figure 3 shows the ML505 Embedded Development Platform. The ML505 has a x1 PCIe connector on one edge of the printed circuit board.

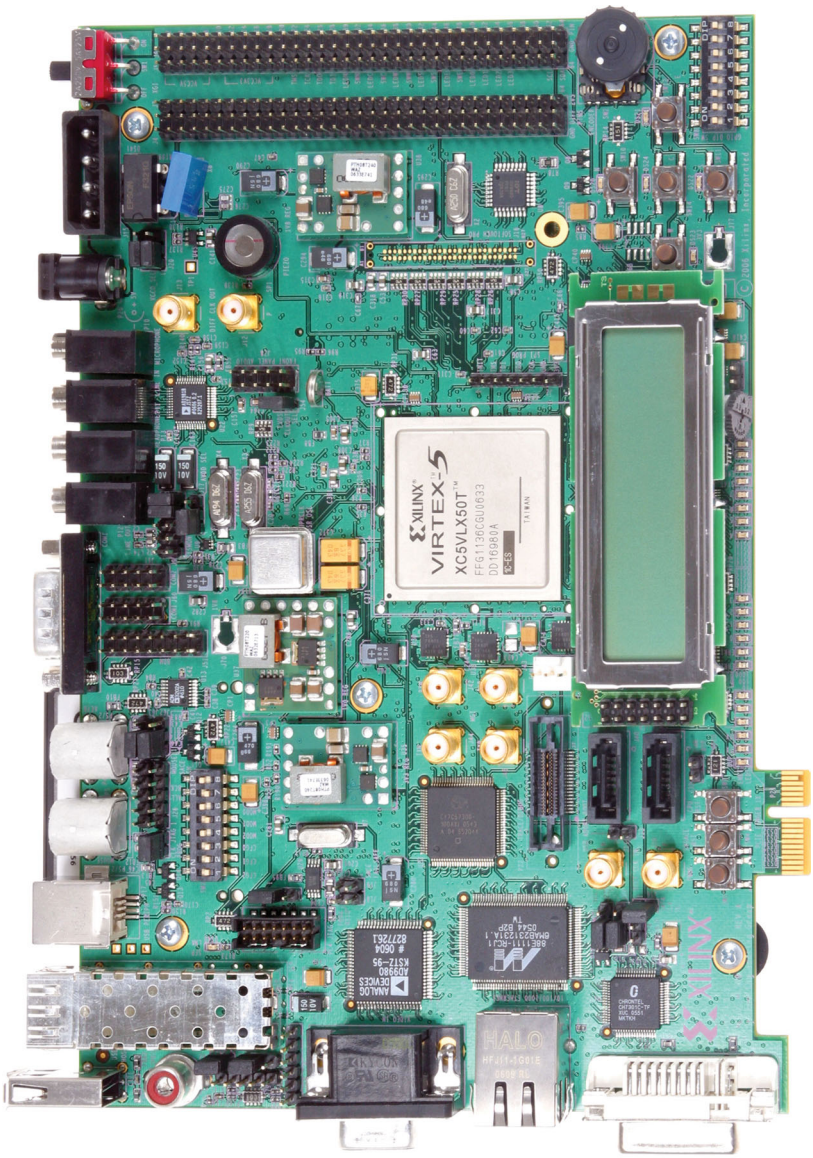


Figure 3: **ML505 PCI/PCIe Evaluation Platform**

Executing the Reference System

The sequence of steps to test the PLBv46 Endpoint Bridge reference system differs depending on whether endpoint to root complex transactions or root complex to endpoint transactions are run. For endpoint to root complex transactions, the steps must be run in the order below. For root complex to endpoint transactions, the steps are the same, but there is no elf to download.

Change directories to the `ready_for_download` directory.

1. Use iMPACT to download the bitstream.

   ```
   impact -batch xapp1030.cmd
   ```

2. Invoke XMD and connect to the MicroBlaze processor.

   ```
   xmd
   ```

```
connect mb mdm
rst
```

3. Download the executable.

```
dow pcie_dma.elf
```

4. Write to the Bridge Control Register to enable Bus Master and the BARs.

```
mwr 0x85C001E0 0x003F0107
```

5. Use the Catalyst to write the PLBv46 Endpoint Bridge Configuration Space Header.

```
File -> Open catalyst/cfg_x1.sdc
```

6. From Catalyst, click

```
Run
```

7. From the XMD prompt, run

```
con
```

Testing the PLBv46 Endpoint Bridge

The system, including the interface to the LeCroy/Catalyst test equipment, is shown in Figure 4. The root complex is the Catalyst or LeCroy test equipment, and the endpoint is the PLBv46 Endpoint Bridge in the ML505 reference system.

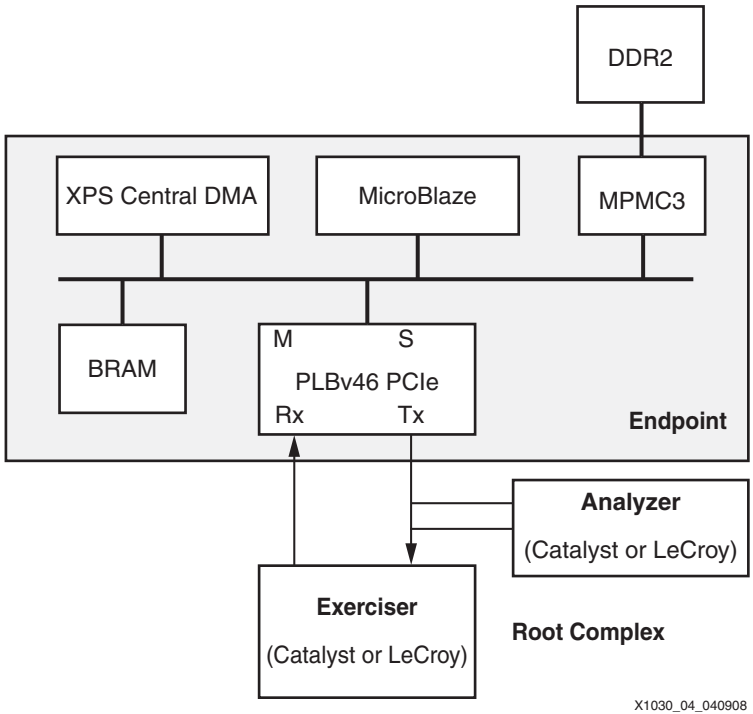


Figure 4: **PLBv46 Endpoint Bridge System Identifying Root Complex/Endpoint**

Endpoint to Root Complex Transactions

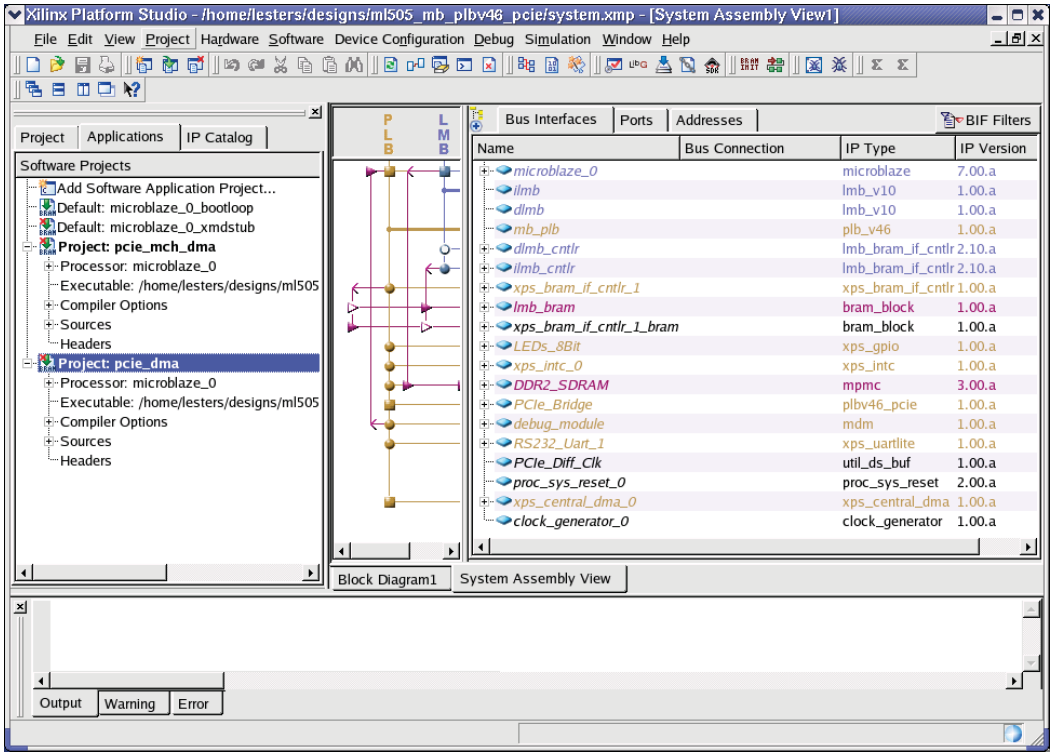
Endpoint to root complex transactions are tested using XMD commands and C code. Two software projects, pcie_dma and pcie_mch_dma, generate Direct Memory Access (DMA) transactions which create PCIe traffic. This code provides an interface to the user which allows the selection of the number of loops to run and the seed. The code generates and verifies pseudo random traffic patterns on the PCIe link.

The `pcie_dma.c` code uses one DMA channel. The pcie_mch_dma.c code allows specification of 1-3 DMA channels.

The PLBv46 Endpoint Bridge Configuration Space Header (CSH) must be written for the code to run correctly. The Catalyst and LeCroy scripts, cfg_x1.sdc and cfg_x1.peg, set up the configuration space header of the PLBv46 Endpoint Bridge.

The Catalyst PCI Express Bus Protocol Exerciser/Analyzer has memory located at address 0x00000000. In the reference systems, the PLBv46 Endpoint Bridge generic C_IPIFBAR2PCIBAR_0 is set to `0x00000000`. This is different from the Base System Builder (BSB) generated value for C_IPIFBAR2PCIBAR_0.

Figure 5 shows the selection of the pcie_dma software project.



X1030_05_040908

Figure 5: **Selecting the pcie_dma Software Project**

pcie_dma. The pcie_dma project runs Direct Memory Access (DMA) operations. The user sets the source address, destination address, and DMA length. The pcie_dma code is used for DMA operations between user defined source and destination addresses. Figure 6 shows the parameters in pcie_dma.c which are edited to test PCI transactions between different memory regions. The elf for pcie_dma.c runs on the MicroBlaze processor in the xc5vlx50t FPGA on the ML505.

pcie_mch_dma. The pcie_mch_dma project runs multi-channel Direct Memory Access (DMA) operations. The user sets the source address, destination address, and DMA length for each channel. The pcie_mch_dma code is used for DMA operations between user defined source and destination addresses. As with the pcie_dma code, the parameters in pcie_mch_dma.c which can be edited to test PCI transactions between different memory regions are `DMAChannel[*].BAR`. The elf for pcie_mch_dma.c, provided in ready_for_download, as pcie_mch_dma.elf, runs on the MicroBlaze processor in the xc5vlx50t FPGA on the ML505.

DMA Transactions

As examples of specifying the source and destination addresses in DMA transactions, the source address may be an address in the ML505 XPS BRAM and the destination address a Catalyst memory across the PCIe link. Another option is source address in Catalyst memory to a second location in Catalyst memory.

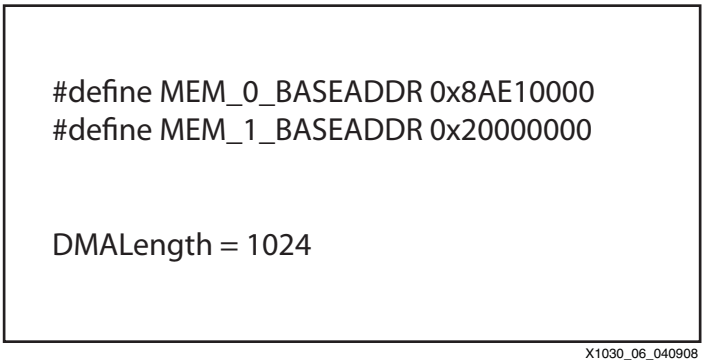


Figure 6: **Defining Source and Destination Addresses, Length in pcie_dma.c**

The XMD scripts and C code generate DMA operations to transfer data between different ML505 and Catalyst memory regions. DMA transactions are generated by writing to the Control, Source Address, Destination Address, and Length registers of the DMA controller. Table 3 provides the register locations for the XPS Central DMA. In the reference design, C_BASEADDR is set to 0x80200000.

Table 3: **XPS Central DMA Registers**

DMA Register	Address
Control Register	C_BASEADDR + 0x04
Source Address Register	C_BASEADDR + 0x08
Destination Address Register	C_BASEADDR + 0x0C
Length Register	C_BASEADDR + 0x10

The pcie_dma.c code consists of the four functions in the functional diagram in Figure 7. The Barberpole Region function provides a rotating data pattern in the memory located at the source address. The Zero Region function sets the memory located at the destination address to all zeroes. The DMA Region function generates a DMA transaction of data located at the source address to the memory at the destination address. Following the DMA transfer, the Verify function verifies that data at the source and destination address are equal.

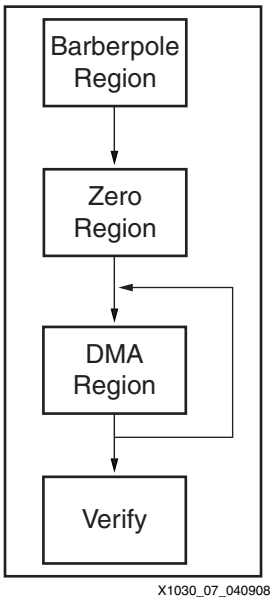


Figure 7: **Functional diagram of pcie_dma.c**

Figure 8 show the communication terminal output when running the pcie_dma/executable.elf.

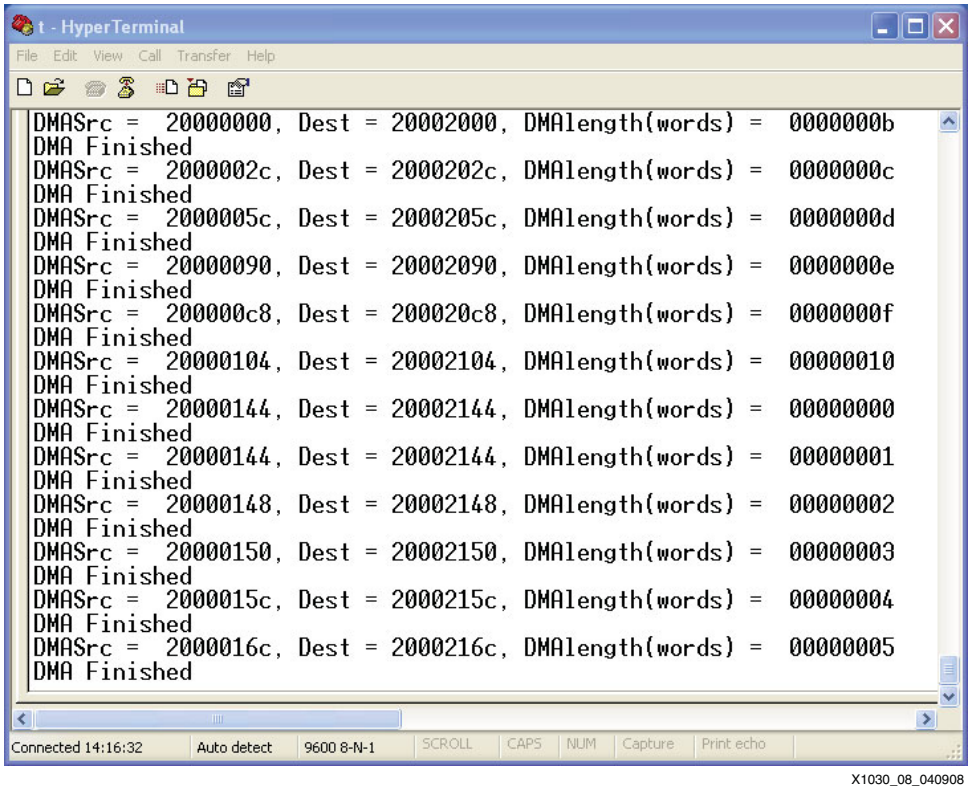

t - HyperTerminal
File Edit View Call Transfer Help
DMASrc = 20000000, Dest = 20002000, DMAlength(words) = 0000000b
DMA Finished
DMASrc = 2000002c, Dest = 2000202c, DMAlength(words) = 0000000c
DMA Finished
DMASrc = 2000005c, Dest = 2000205c, DMAlength(words) = 0000000d
DMA Finished
DMASrc = 20000090, Dest = 20002090, DMAlength(words) = 0000000e
DMA Finished
DMASrc = 200000c8, Dest = 200020c8, DMAlength(words) = 0000000f
DMA Finished
DMASrc = 20000104, Dest = 20002104, DMAlength(words) = 00000010
DMA Finished
DMASrc = 20000144, Dest = 20002144, DMAlength(words) = 00000000
DMA Finished
DMASrc = 20000144, Dest = 20002144, DMAlength(words) = 00000001
DMA Finished
DMASrc = 20000148, Dest = 20002148, DMAlength(words) = 00000002
DMA Finished
DMASrc = 20000150, Dest = 20002150, DMAlength(words) = 00000003
DMA Finished
DMASrc = 2000015c, Dest = 2000215c, DMAlength(words) = 00000004
DMA Finished
DMASrc = 2000016c, Dest = 2000216c, DMAlength(words) = 00000005
DMA Finished
Connected 14:16:32
Auto detect
9600 8-N-1
SCROLL
CAPS
NUM
Capture
Print echo
X1030_08_040908

Figure 8: **pcie_dma.c output**

Catalyst Testing

This section discusses testing using Catalyst Enterprises SPX Series PCI Express Analyzer/Exerciser system. The SPX is a serial bus Analyzer/Exerciser used to analyze and/or exercise PCI Express data transactions. The SPX4 Analyzer consists of the SPX4 card and Analyzer software. The Analyzer allows capture and trigger on Transaction and Data Link Layer Packets, Physical Layer Ordered Sets, and all bus conditions. The Exerciser generates bus traffic while operating as either a root complex or endpoint device.

Figure 9 shows a functional diagram of the Catalyst test setup.

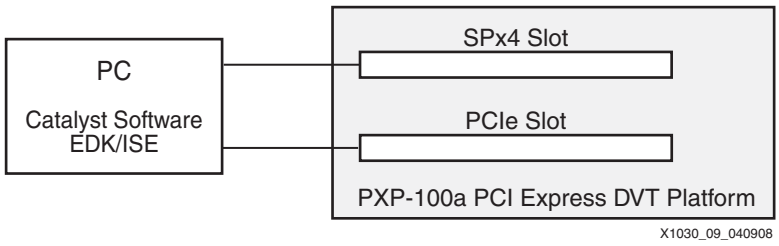


Figure 9: **Catalyst Test Setup**

Figure 10 is a photograph of the Catalyst setup. The ML505 is inserted into the PCIe slot. The Platform Cable USB cable is connected to the ML505 to use Impact, XMD, and GDB. A USB cable connects the PC based Catalyst software to the SPX4 Analyzer.

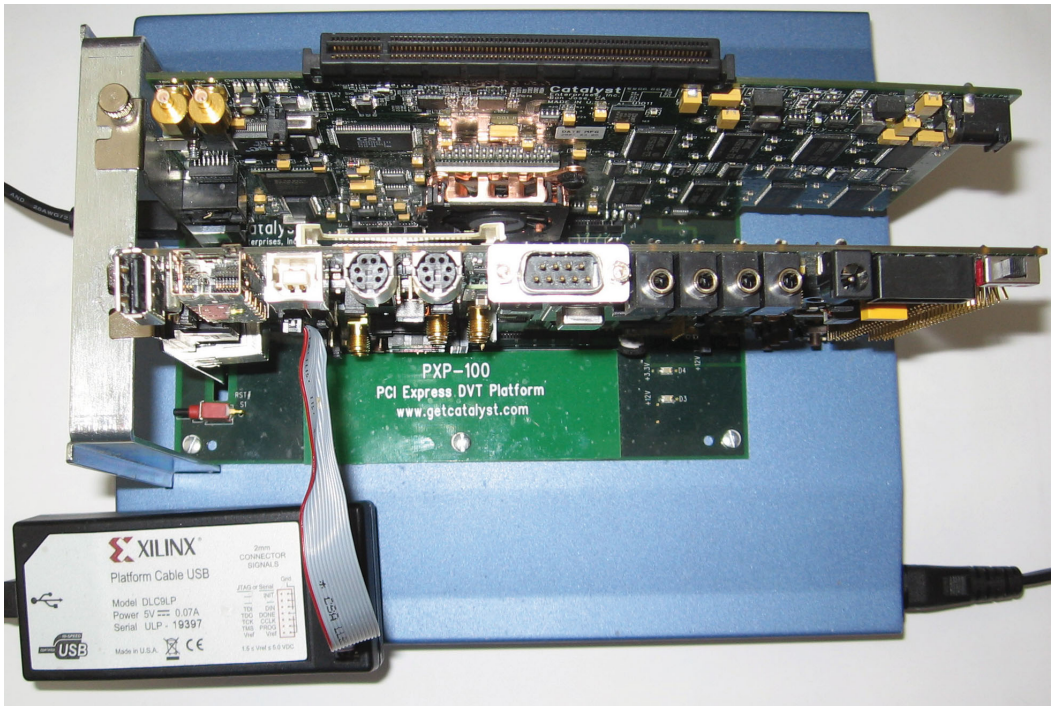


Figure 10: **Photo of Catalyst PCI Express Test Equipment**

In addition to using the Catalyst Bus Protocol Analyzer/Exerciser software as discussed extensively in this application note, the Catalyst SpekChekTM PCI Express Compliance Suite has been run with this reference design to verify that the PLBv46 Endpoint Bridge meets PCI SIG compliance tests. The SpekChek tests are defined in the SpekChek User Manual Version 6.5.

After downloading the bit file into the ML505 FPGA using Impact, the PLBv46 Endpoint Bridge Bridge Control Register (BCR) is written as shown in Figure 11. The BCR enables the PCIe Bus Master and the Base Address Registers (BARs). This step must be done prior to using the Catalyst software.

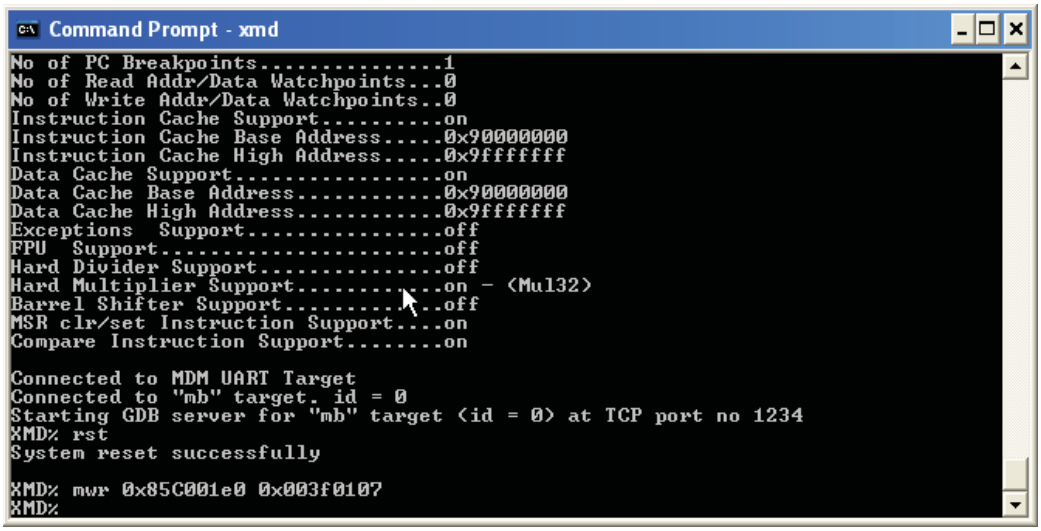

Command Prompt - xmd
No of PC Breakpoints.....1
No of Read Addr/Data Watchpoints...0
No of Write Addr/Data Watchpoints..0
Instruction Cache Support.....on
Instruction Cache Base Address.....0x90000000
Instruction Cache High Address.....0x9fffffff
Data Cache Support.....on
Data Cache Base Address.....0x90000000
Data Cache High Address.....0x9fffffff
Exceptions Support.....off
FPU Support.....off
Hard Divider Support.....off
Hard Multiplier Support.....on - (Mul32)
Barrel Shifter Support.....off
MSR clr/set Instruction Support....on
Compare Instruction Support.....on
Connected to MDM UART Target
Connected to "mb" target. id = 0
Starting GDB server for "mb" target (id = 0) at TCP port no 1234
XMD% rst
System reset successfully
XMD% mwr 0x85C001e0 0x003f0107
XMD%
X1030_11_040908

Figure 11: **Writing the Bridge Control Register**

Five tabs are used to setup the Catalyst PCIe Bus Protocol Analyzer/Exerciser. Figure 12 shows Catalyst **Capture** settings. The option selected is to **Capture Everything except Idles**. In the **Trigger On** tab, select **Pattern** and **Trigger on TLP (Any Type)**. Select **Any Direction**. In the **Settings** tab, specify the name of the output ssf file.

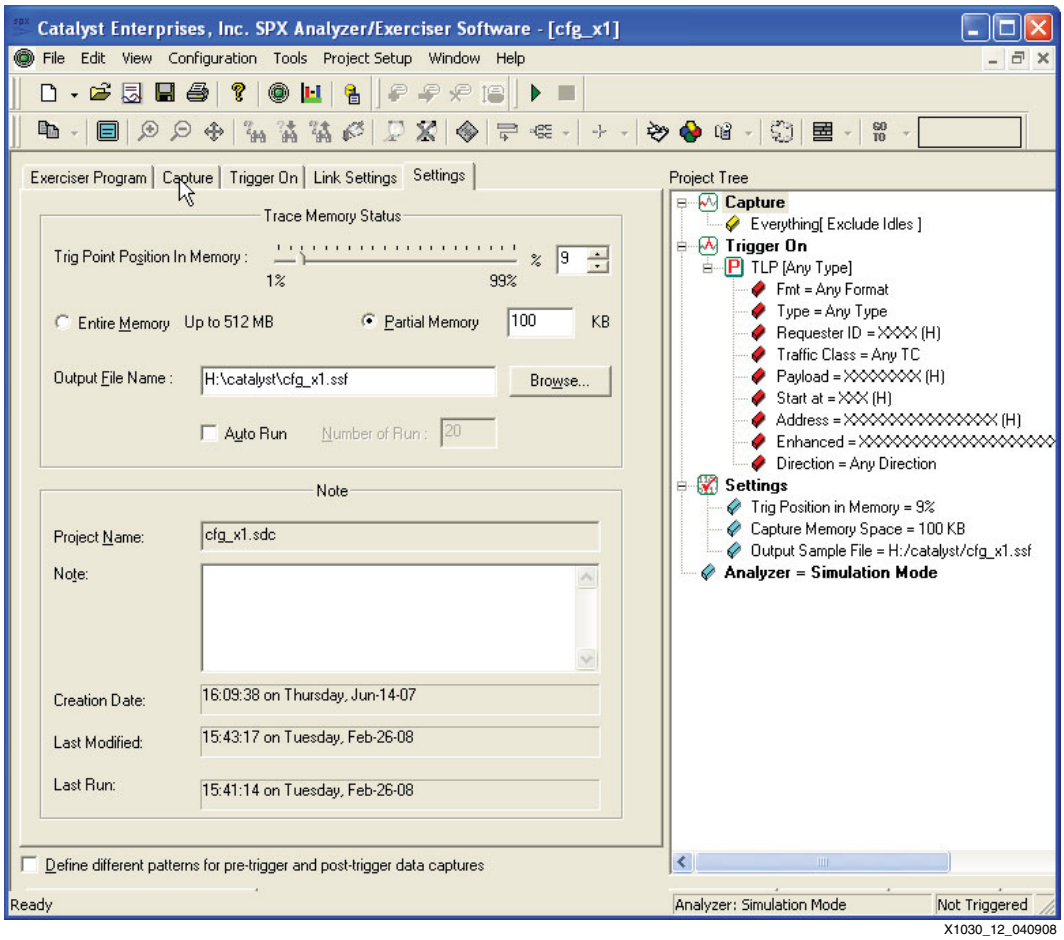


Figure 12: **Capture Settings**

Figure 13 shows the setup of the Catalyst **Link Settings**. Select the Platform mode (hidden behind the Link Status pane). Click on the Link Status button to invoke the Link Status pane displayed. The figure shows a Link Width = 1, so the link is up and trained as x1.

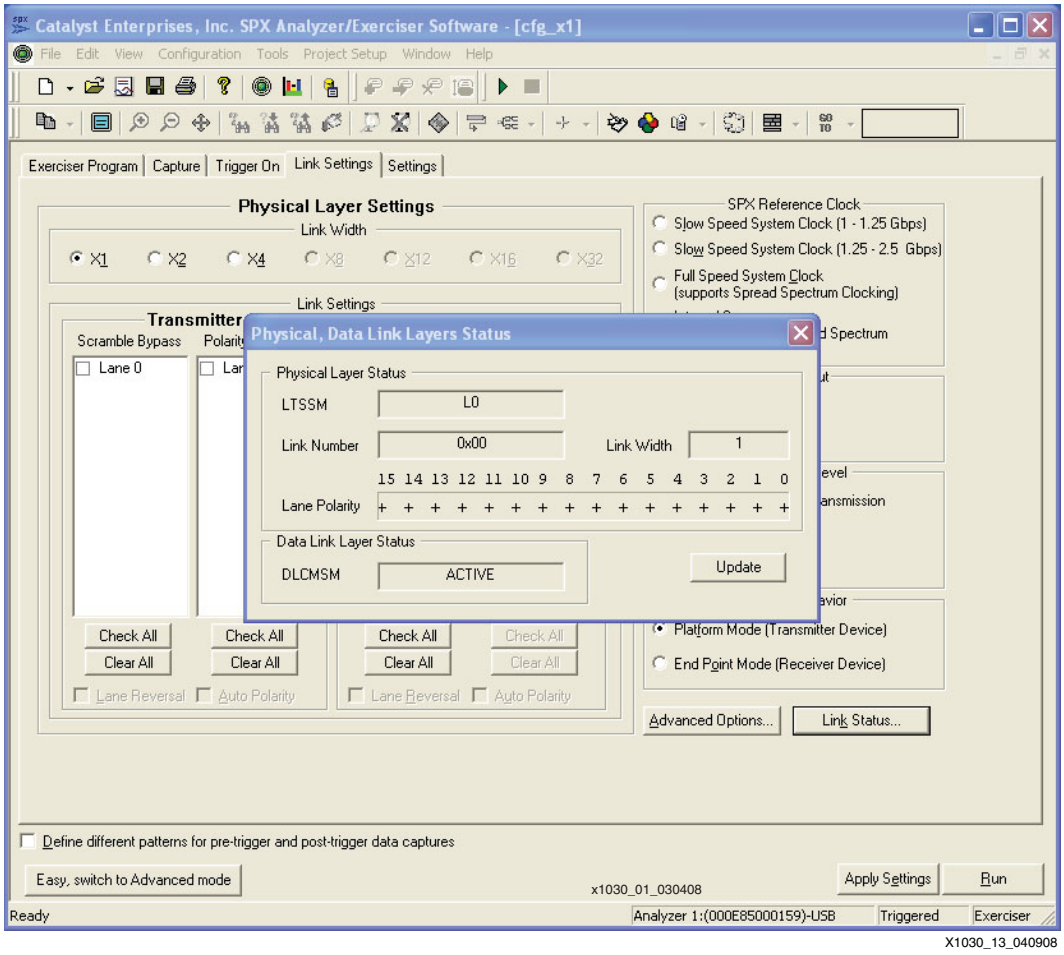


Figure 13: **Catalyst Link Settings**

Figure 14 is a graphical view of the stimuli for configuring the PLBv46 Endpoint Bridge, including BAR 0. The `ml505_mb_plbv46_pcie/catalyst` directory contains the cfg_x1.sdc stimuli file. The cfg_x1.sdc project is loaded using the File -> Open pull down menu. The *.sdc files are readable text files which contains the transactions used as stimuli.

In cfg_x1.sdc, the Device ID/Vendor ID is read. The Command Status register is written and read. The Revision ID and Class Code register is read.

In the figure, the Name column provides the type of transaction and the Reg Num column specifies the register in the Configuration Space Header.

BAR0 is written and read. BAR0 is a 64-bit BAR with the lower 32-bits defined at Configuration Space Header (CSH) Register Number 4 and the higher 32-bits defined at CSH Register Number 5.

Packets 10 and 11 are Configuration Writes and packets 12 and 13 are Configuration Reads.

In the Data field in packet 10, the endianess of the data written is swapped.

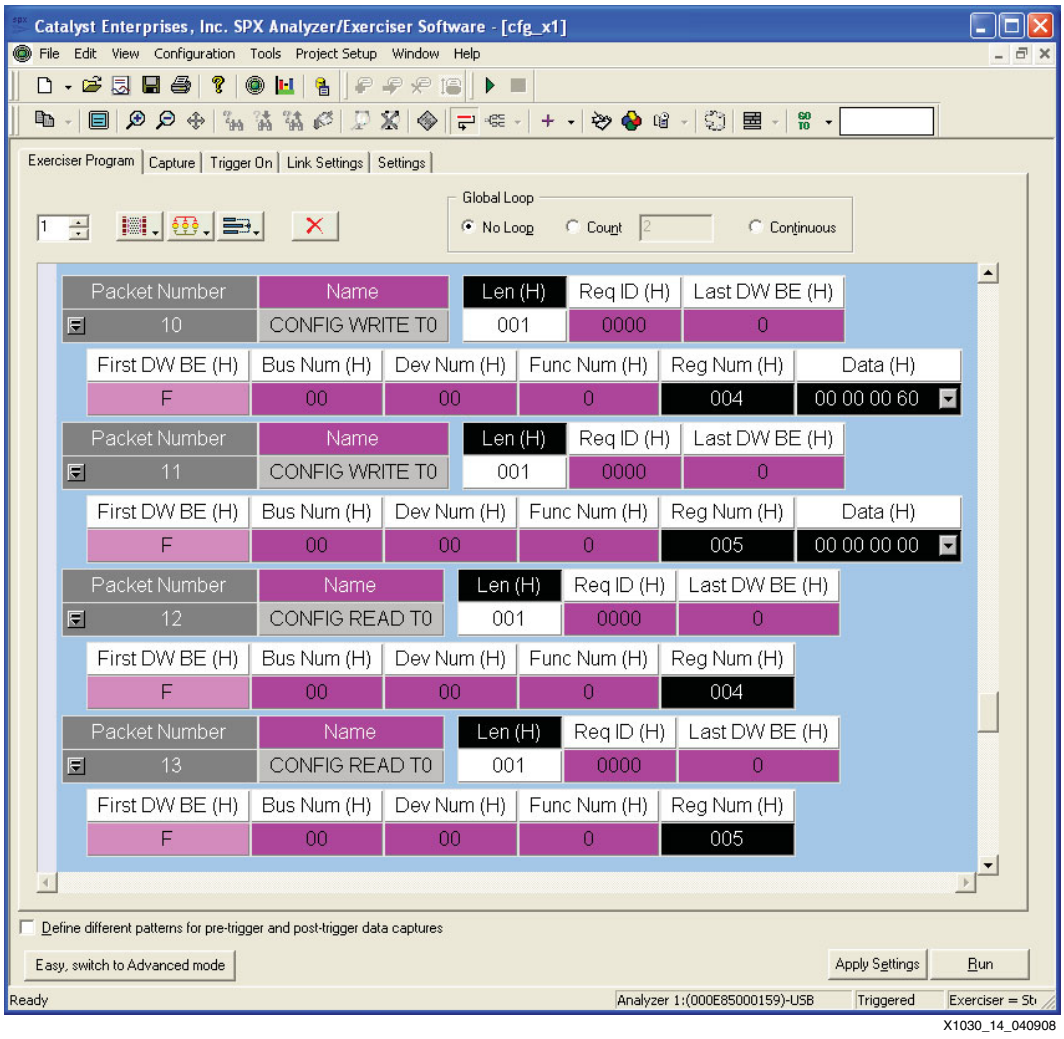

Catalyst Enterprises, Inc. SPX Analyzer/Exerciser Software - [cfg_x1]
File Edit View Configuration Tools Project Setup Window Help
Exerciser Program Capture Trigger On Link Settings Settings
Global Loop
No Loop Count Continuous
Packet Number Name Len (H) Req ID (H) Last DW BE (H)
10 CONFIG WRITE T0 001 0000 0
First DW BE (H) Bus Num (H) Dev Num (H) Func Num (H) Reg Num (H) Data (H)
F 00 00 0 004 00 00 00 60
Packet Number Name Len (H) Req ID (H) Last DW BE (H)
11 CONFIG WRITE T0 001 0000 0
First DW BE (H) Bus Num (H) Dev Num (H) Func Num (H) Reg Num (H) Data (H)
F 00 00 0 005 00 00 00 00
Packet Number Name Len (H) Req ID (H) Last DW BE (H)
12 CONFIG READ T0 001 0000 0
First DW BE (H) Bus Num (H) Dev Num (H) Func Num (H) Reg Num (H)
F 00 00 0 004
Packet Number Name Len (H) Req ID (H) Last DW BE (H)
13 CONFIG READ T0 001 0000 0
First DW BE (H) Bus Num (H) Dev Num (H) Func Num (H) Reg Num (H)
F 00 00 0 005
Define different patterns for pre-trigger and post-trigger data captures
Easy, switch to Advanced mode
Apply Settings
Run
Ready
Analyzer 1:(000E85000159)-USB
Triggered
X1030_14_040908

Figure 14: **Catalyst Configuration Stimuli**

Figure 15 shows the Analyzer output after running cfg_x1. The results are contained in the cfg_x1.ssf file. Registers in the Configuration Space Header are displayed in packet 0 using Vendor ID and Device ID symbolic names, with Xilinx 0x10EE and 0x0505 values. The Command Status Register is read. The SC in the status field indicates successful completion of the transaction. In the figure, the Revision ID and Class Code Register field is expanded to provide a readable table of the values in the Data field.

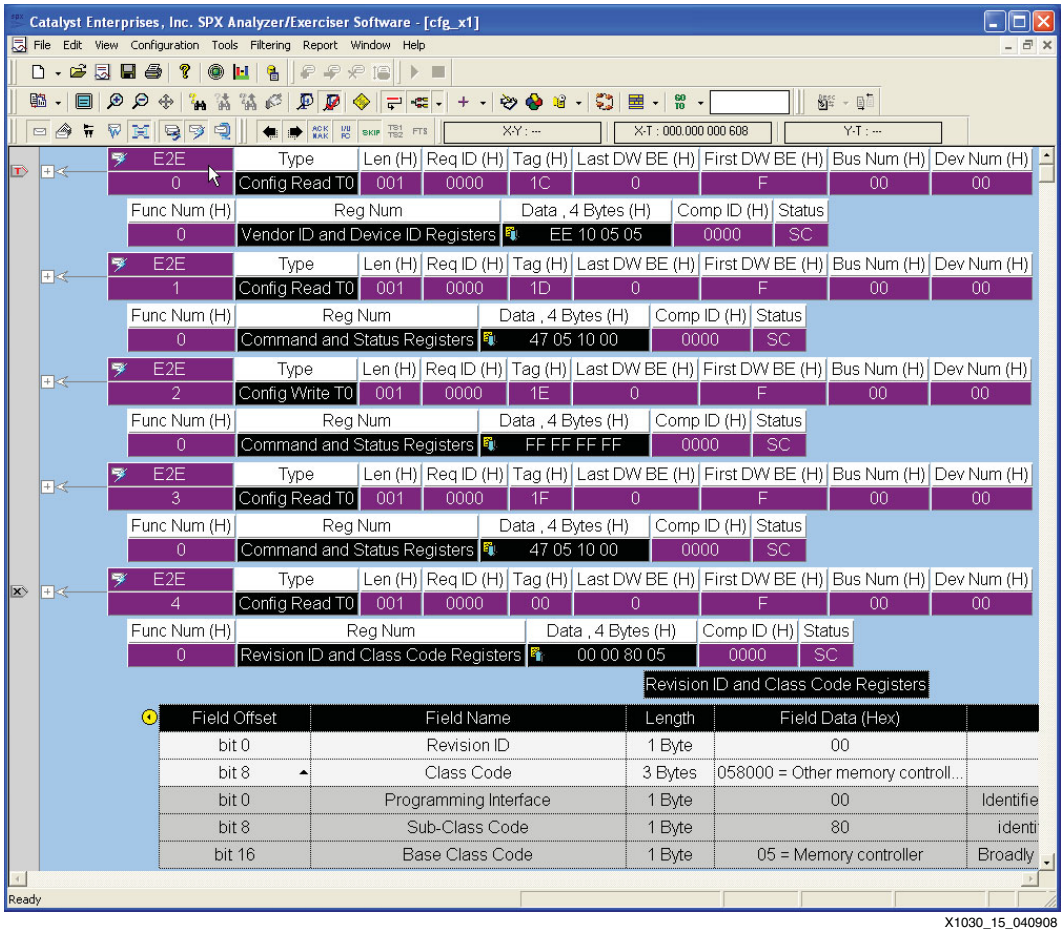

Catalyst Enterprises, Inc. SPX Analyzer/Exerciser Software - [cfg_x1]
File Edit View Configuration Tools Filtering Report Window Help
X-Y : ---
X-T : 000.000 000 608
Y-T : ---
E2E | Type | Len (H) | Req ID (H) | Tag (H) | Last DW BE (H) | First DW BE (H) | Bus Num (H) | Dev Num (H)
0 | Config Read T0 | 001 | 0000 | 1C | 0 | F | 00 | 00
Func Num (H) | Reg Num | Data , 4 Bytes (H) | Comp ID (H) | Status
0 | Vendor ID and Device ID Registers | EE 10 05 05 | 0000 | SC
E2E | Type | Len (H) | Req ID (H) | Tag (H) | Last DW BE (H) | First DW BE (H) | Bus Num (H) | Dev Num (H)
1 | Config Read T0 | 001 | 0000 | 1D | 0 | F | 00 | 00
Func Num (H) | Reg Num | Data , 4 Bytes (H) | Comp ID (H) | Status
0 | Command and Status Registers | 47 05 10 00 | 0000 | SC
E2E | Type | Len (H) | Req ID (H) | Tag (H) | Last DW BE (H) | First DW BE (H) | Bus Num (H) | Dev Num (H)
2 | Config Write T0 | 001 | 0000 | 1E | 0 | F | 00 | 00
Func Num (H) | Reg Num | Data , 4 Bytes (H) | Comp ID (H) | Status
0 | Command and Status Registers | FF FF FF FF | 0000 | SC
E2E | Type | Len (H) | Req ID (H) | Tag (H) | Last DW BE (H) | First DW BE (H) | Bus Num (H) | Dev Num (H)
3 | Config Read T0 | 001 | 0000 | 1F | 0 | F | 00 | 00
Func Num (H) | Reg Num | Data , 4 Bytes (H) | Comp ID (H) | Status
0 | Command and Status Registers | 47 05 10 00 | 0000 | SC
E2E | Type | Len (H) | Req ID (H) | Tag (H) | Last DW BE (H) | First DW BE (H) | Bus Num (H) | Dev Num (H)
4 | Config Read T0 | 001 | 0000 | 00 | 0 | F | 00 | 00
Func Num (H) | Reg Num | Data , 4 Bytes (H) | Comp ID (H) | Status
0 | Revision ID and Class Code Registers | 00 00 80 05 | 0000 | SC
Revision ID and Class Code Registers
Field Offset | Field Name | Length | Field Data (Hex)
bit 0 | Revision ID | 1 Byte | 00
bit 8 | Class Code | 3 Bytes | 058000 = Other memory controll...
bit 0 | Programming Interface | 1 Byte | 00 Identifie
bit 8 | Sub-Class Code | 1 Byte | 80 identi
bit 16 | Base Class Code | 1 Byte | 05 = Memory controller Broadly
Ready
X1030_15_040908

Figure 15: **Results from Catalyst Configuration**

Figure 16 shows an excerpt of the Exerciser cfg_x1.sdc file. The file contains the stimuli TLPs. While it is generally easier to read and edit the TLPs using the Catalyst Display Viewer, the text file is readable and editable, and more details are provided than can be efficiently presented in a Display Viewer. The figure shows the content of a single Configuration Read TLP.

```
Packet_Type = "Config Read T0"
Framing_Symbol1 = "FB"
Reserved_1 = "0"
Sequence_Number = "000"
Reserved_2 = "0"
Format = "0"
Type = "04"
Reserved_3 = "0"
TC = "0"
Reserved_4 = "0"
TD = "0"
EP = "0"
Attribute = "0"
Reserved_5 = "0"
Length = "001"
Requester_ID = "0000"
Tag = "00"
Last_DW_BE = "0"
First_DW_BE = "F"
Bus_Number = "00"
Device_Number = "00"
Function_Number = "0"
Reserved_6 = "0"
Register_Address = "000"
Reserved_7 = "0"
TLP_Digest = ""
LCRC = "2AC19647"
Framing_Symbol2 = "FD"
Loop_Type = "No_Loop"
Loop_Count = ""
Iterate_After_Trigger = "No"
Delay_Count = "0"
Trigger_Source = "Immediate_Execution"
Disparity_Error = "No"
ZData = "10000000000000000001"
Symbol_View = "Collapse"
Trigger_Output = "No"
Trigger_Output_Type = "Pulse"
Global_Loop
```

X1030_16_040908

Figure 16: **sdc_example**

As Root Complex, the Catalyst Exerciser performs memory writes and memory reads to the ML505 memory. The ML505 reference design contains XPS BRAM and a Multiported Memory Controller (MPMC) interface to DDR2. Figure 17 shows the memory addressing for Root Complex (Catalyst) to Endpoint (ML505) transactions.The memory addressed is controlled by the BAR value written and by the C_PCIBAR2IPIFBAR_* generic(s). In the reference design, BAR0 is written as 0x0000000060000000. C_PCIBAR2IPIFBAR_0 addresses XPS BRAM at 0x8AE10000, and C_PCIBAR2IPIFBAR_1 addresses DDR2 at location 0x90000000.

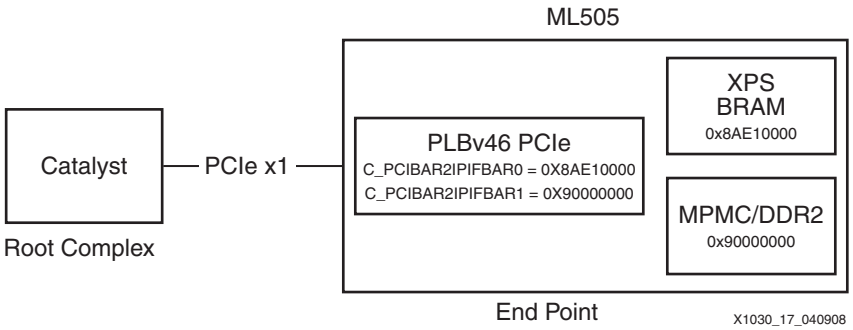


Figure 17: **Catalyst Root Complex**

Figure 18 shows the write then read TLPs in the wr_rd_x1.sdc file. In the figure, Packet 0 is a MWr64 to address 0x0000000060000000 of 128 bytes. The Data Field allows the user to specify data as Upcount, Walking Bit, or Random pattern, or a user defined pattern such as 0x12345678 can be entered. As exercises in learning to use the PLBv46 Endpoint Bridge, the data can be varied, and the memory written/read can be changed from XPS BRAM to DDR2.

The Length field is 020H which is 32 doublewords (DWs) or 128 bytes.

Packet 1 is a MRd64 of address 0x0000000060000000, used to verify the written data. The MRd64 TLP address endianess differs from the CfgWr address endiness used when the BAR was written with a CfgWr in Figure 14. Bit Order and Endianess can be defined by right clicking a field to invoke a pop up menu.

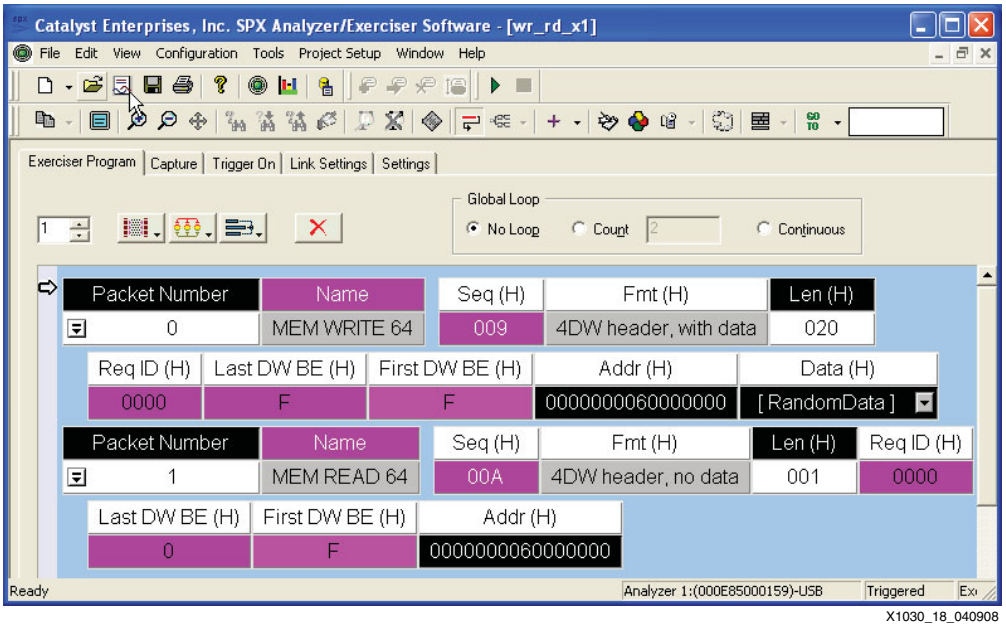


Figure 18: **wr_rd_x1 TLP stimuli**

Figure 19 shows the results after running a version of wr_rd_x1.sdc in which a random pattern of 0x0AADC5B9F1B0DC89 is transmitted.

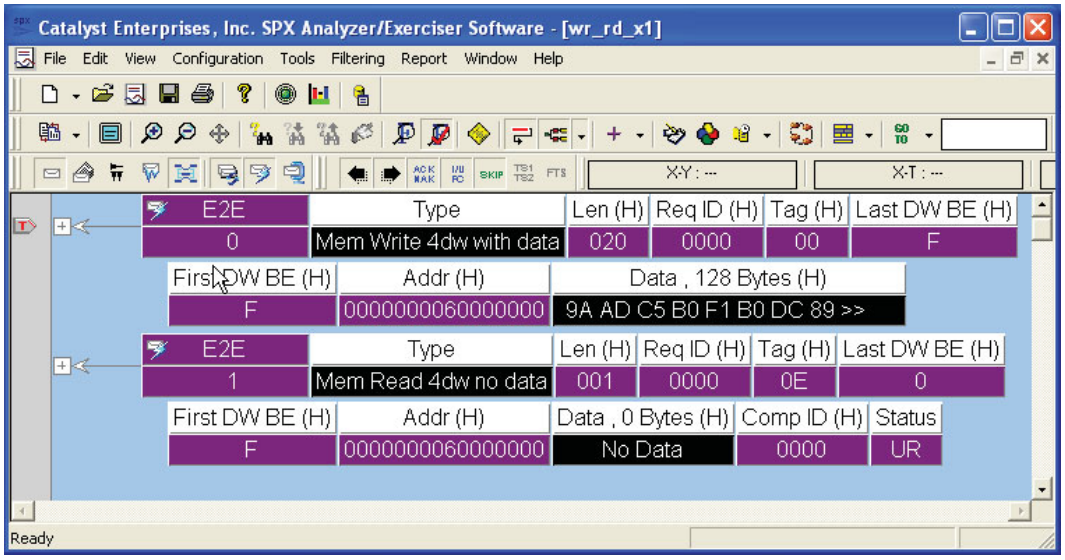


Figure 19: **Catalyst wr_rd_x1 Results**

Figure 20 shows the use of XMD to read ML505 DDR2 memory to provide a second verification that the wr_rd_x1.sdc script functioned as intended. The data read in XMD should be the same as the data in the Analyzer waveform display.

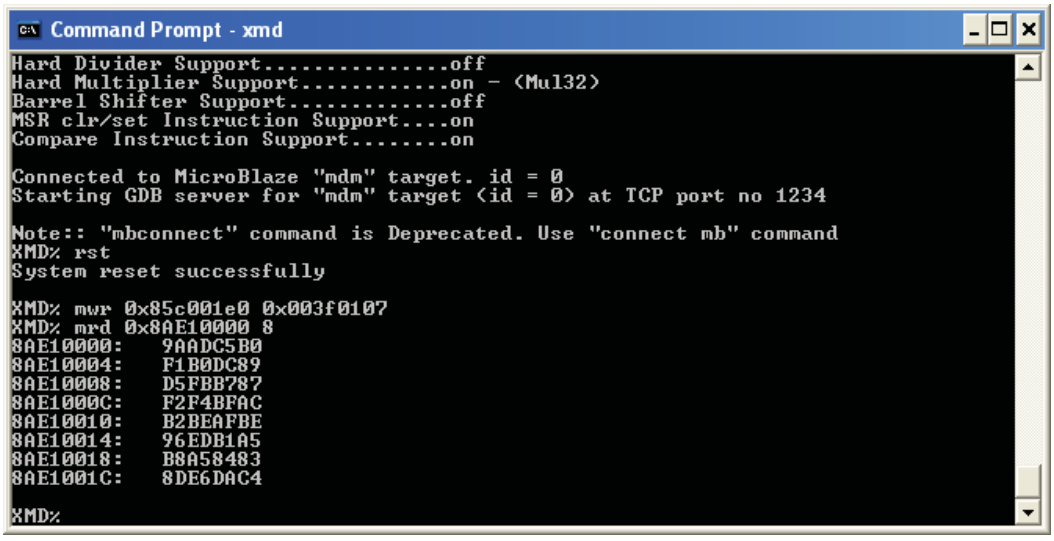


Figure 20: **Verifying Root Complex to Endpoint Transactions with XMD**

Using Catalyst to test PCIe Performance

Catalyst is used for performance testing. This section provides performance tests for Root Complex to Endpoint transactions, first for read transactions and then for write transactions. The test setup is defined and then performance results are given for various lengths for 32 and 64 bit transactions.

Figure 21 shows the physical link setup for the performance test. For the ml505_mb_plbv46_pcie project, change the Physical Layer Settings Link Width to x1.

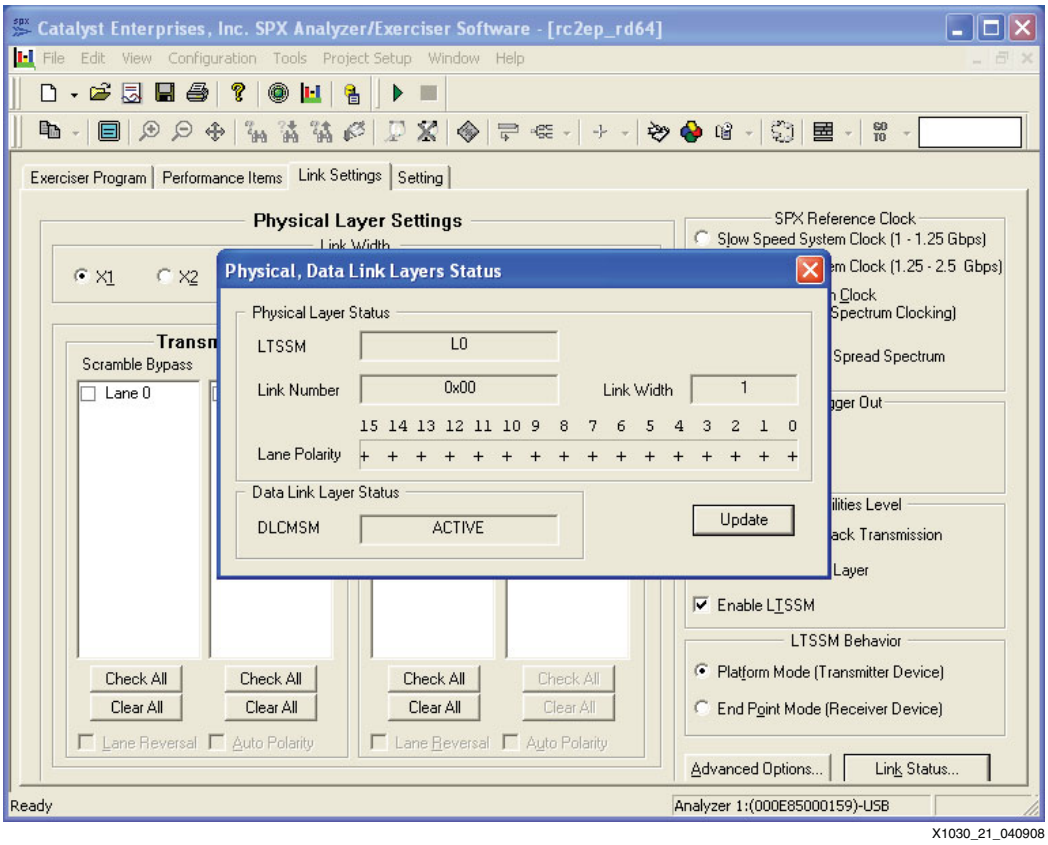


Figure 21: **Performance Test Physical Settings**

Root Complex to Endpoint Performance Tests

To setup the performance test, the ML505 is inserted into the Catalyst. The bitstream is downloaded into the FPGA. Use XMD to write 0x003F0107 to the PLBv46 Endpoint Bridge Bridge Control Register to enable the Bus Master and BARs.

Root Complex to Endpoint Read Operations

Figure 22 shows the opening of the rc2ep_rd64 performance project. Performance projects use the spf extension.

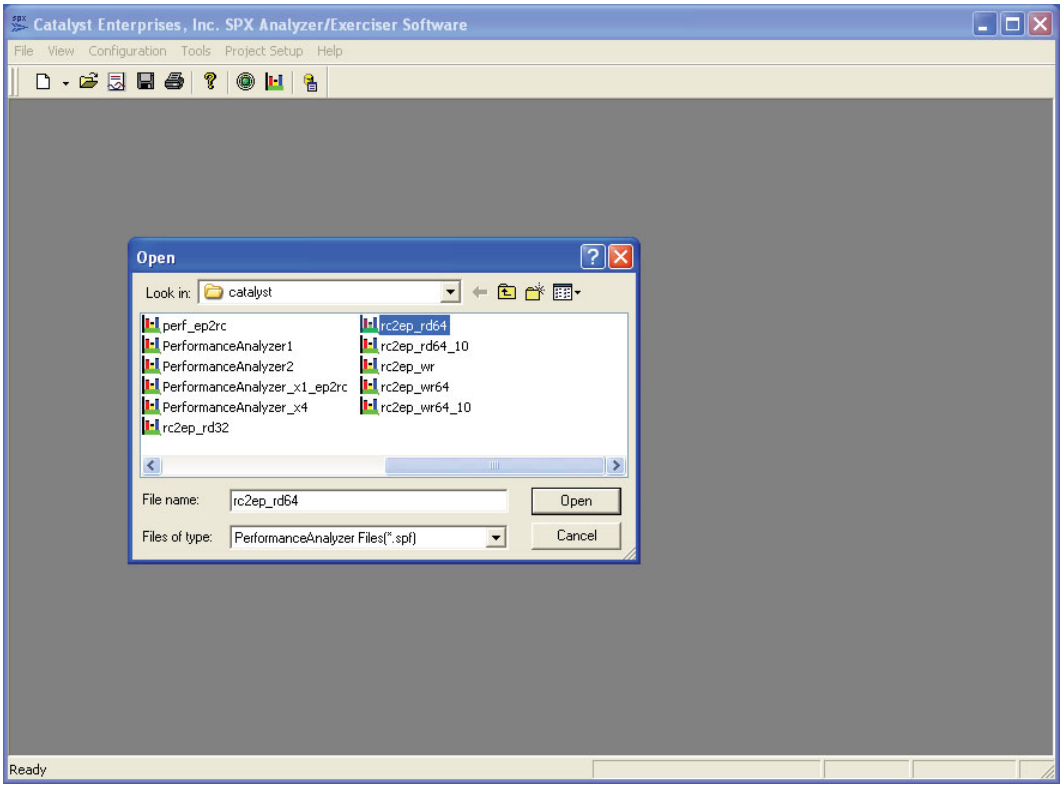


Figure 22: **Opening a Catalyst Performance Test**

The four tabs used in performance projects are the Exercise Program, Performance Items, Link Settings, and Settings. In Performance Items, the type of performance tests run are defined. The PCIe traffic used in the performance measurement is defined in the Exercise Program.

Figure 23 shows a single TLP used in the performance measurements of Rd64 transactions of length = 003. Click the TLP button below Performance Items to add the TLP to the Exercise Program. Using the pop up menu, select Memory -> Read Request 64 bits. Fill out the address and Len fields. Select the **Continuous** radio button so that the TLP is continuously transmitted.

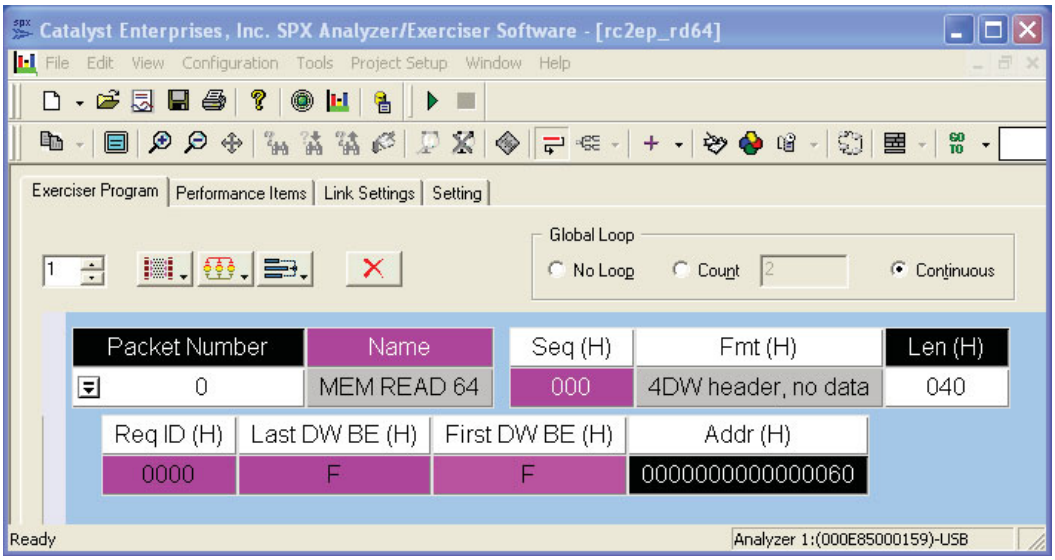


Figure 23: **Defining MRd64 Performance Stimuli**

The next figures show the performance results of MRd64 transactions, varying the length of the TLP. The single continuously transmitted TLP stimuli just defined is shown in the pane at the bottom of the figure. The left pane is a Link Chart which provides the average payload size. The right pane is a Link Chart which provides the data throughput and the payload throughput.

In the **Performance Items** tab, **Link Usage**, **Number of Packets**, and **Latency** are unchecked. Under **Report Directions**, **Aggregate** is checked.

In the following tests, Data Throughput is the overall bus traffic of all non-idle packets divided by the update interval. Payload Throughput is the payload data of TLPs divided by the update interval. The update interval, defined in the Settings tab, for the performance measurements in this document is 1 second.

The MRd performance is the round trip time including the MRd and Completiion with Data packets.

Figure 24 shows the performance results of a MRd64 TLP of length 10. The data and payload throughput are 135.1 MB/s and 133.5 MB/s.

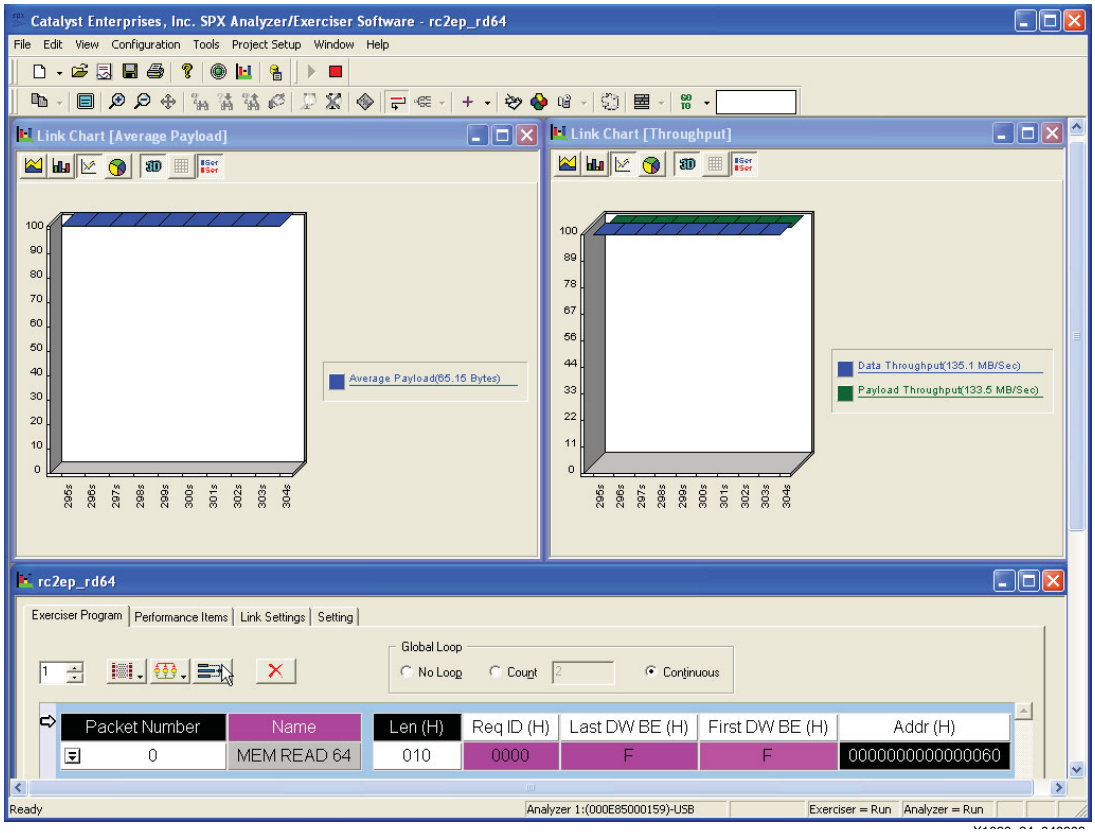


Figure 24: **MRd64 Performance Results - Length = 10**

Figure 25 shows the performance results of a MRd64 TLP of length 100. The data and payload throughput are 135.2 MB/s and 133.6 MB/s.

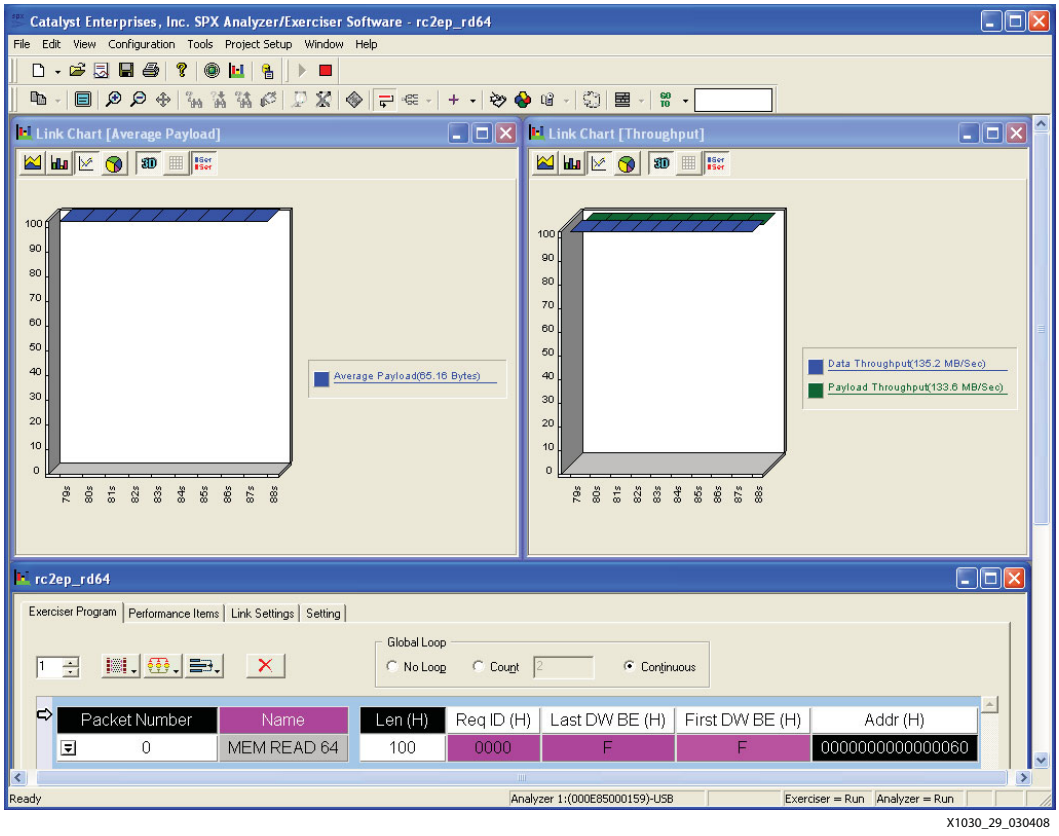


Figure 25: **MRd64 Performance Results - Length = 100**

Because of a Catalyst limitation, performance measurements can not be done for TLPs longer than 400 bytes.

Figure 26 shows the performance of MRd32 transactions of length = 3.The data and payload throughput are 114.6 MB/s and 15.4 MB/s.

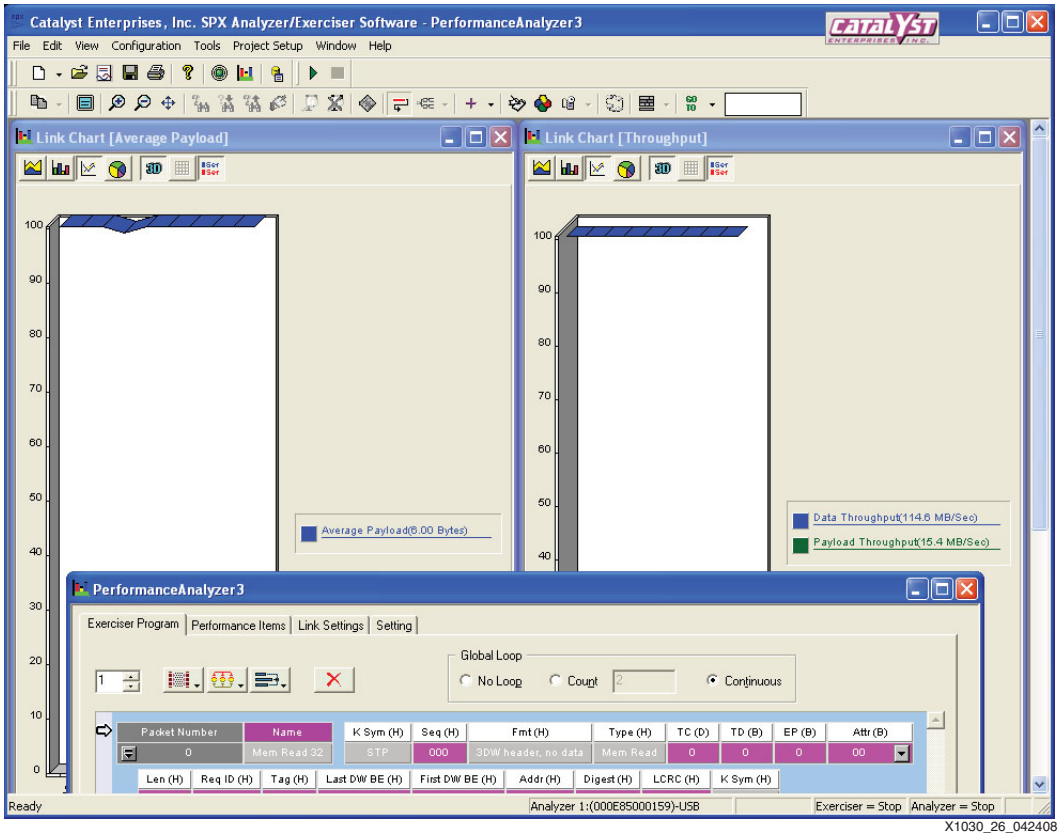
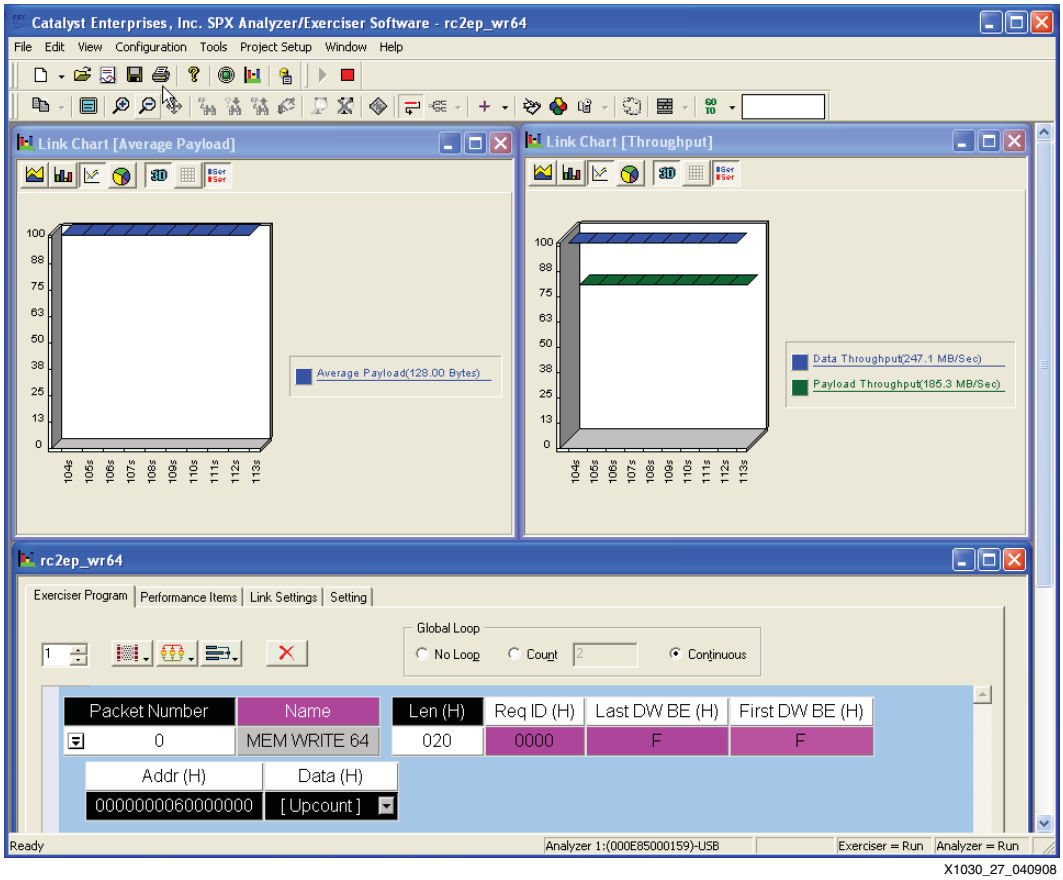


Figure 26: **MRd32 Performance Results - Length = 3**

Root Complex to Endpoint Write Transactions

Figure 27 shows a write transaction. The length field is set to 020H or 128 bytes. The data written is an Upcount pattern. The Continuous radio button is selected.



X1030_27_040908

Figure 27: **MWr64 Performance Results, Length = 20**

Figure 28 shows the performance results from running a continuous MWr32 transaction. The data and payload throughput are 241.3 MB/s and 185.3 MB/s.

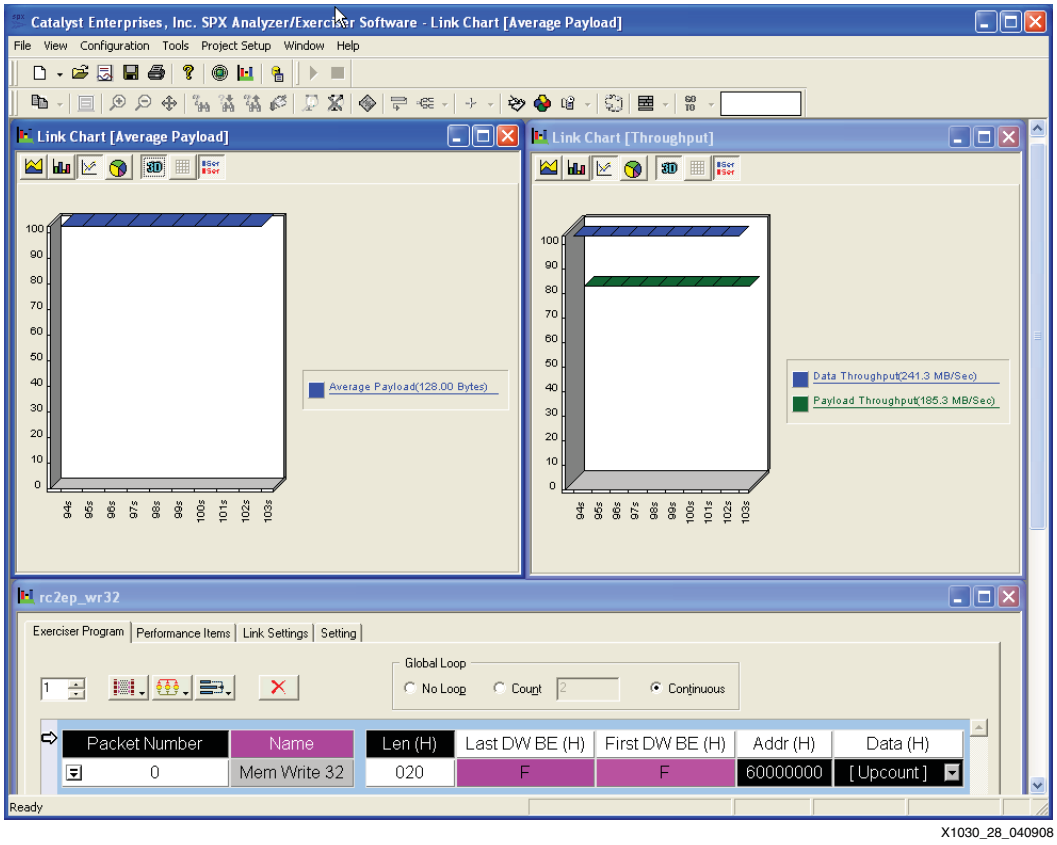


Figure 28: **MWr32 Performance Results, Length = 20**

Endpoint to Root Complex Transactions

This section measures the performance of Endpoint to Root Complex transactions. The stimuli for these transactions are generated using the Xilinx XPS Central DMA Controller (DMAC) in the system.mhs. The functionality of the DMA controller is discussed earlier in this application note. The DMA transaction is from the address specified in the DMAC Source Address register to the address specified in the DMAC Destination Address register. The length of the DMA transaction is specified by the value in the DMAC Length register.

Prior to generating the stimuli, the performance test is set up. Figure 29 shows the importing of the performance test setup file `catalyst/pcie_dma.spf`. The throughput measurements are aggregate.

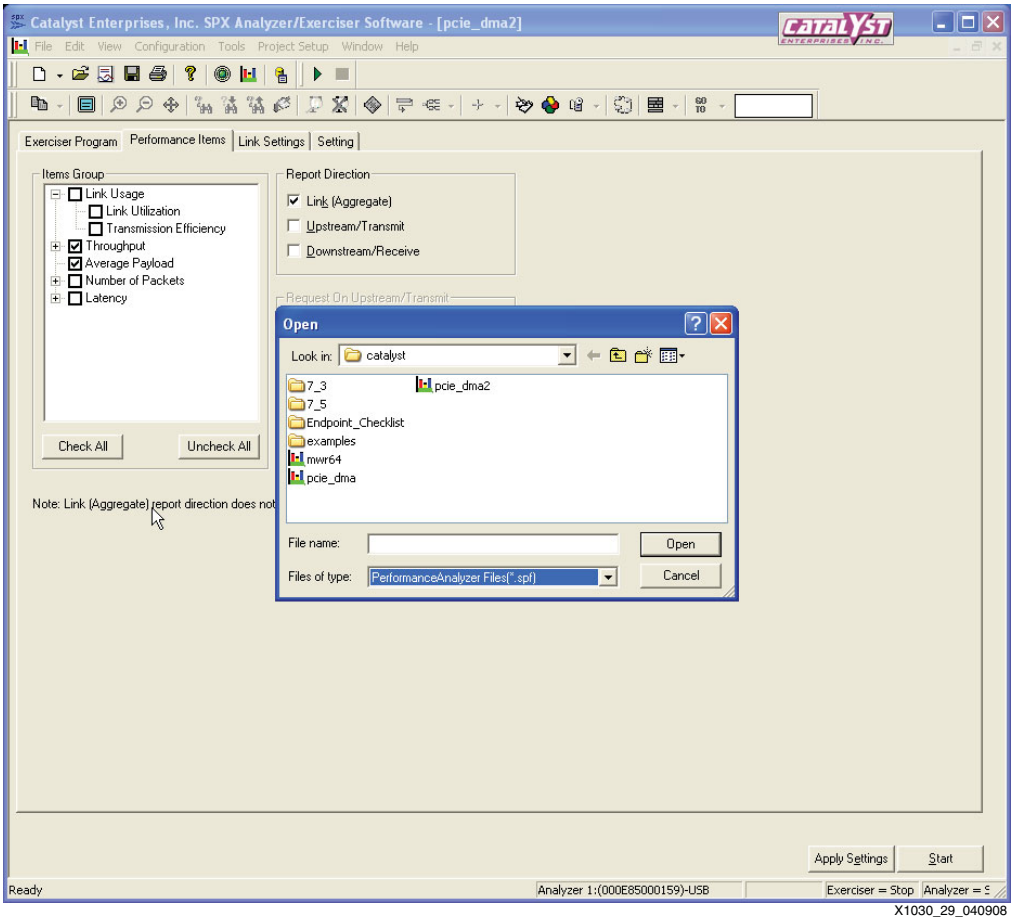


Figure 29: **Importing Performance Test Setup**

To generate stimuli, either C code or an XMD script is used to write the DMAC registers. Figure 30 shows an XMD script to generate stimuli. Using XMD scripts and commands allows the relatively quick verification that the operation is functioning correctly. After running a DMA transaction, a mrd command can verify that the data at the source and destination addresses are equivalent. XMD commands may be too slow to give maximum performance results.

The DMA Status Register is monitored to determine if the DMAC is Busy. When it is not busy, a DMA transaction is initiated by a write to the DMAC Length register.

```
set outfile [open "dma.txt" "w"]
connect mb  mdm
rst

puts $outfile [mwr 0x85C001E0 0x003F0107]

puts $outfile [mwr 0x80200000 0x0000000A]
puts $outfile [mwr 0x80200030 0x00000003]
puts $outfile [mwr 0x80200004 0xC0000004]
puts $outfile [mwr 0x80200008 0x20000000]
puts $outfile [mwr 0x8020000C 0x20002000]


puts $outfile [mwr 0x20000000 0x12345678 100]
puts $outfile [mwr 0x20002000 0x0 100]

set DMASR [mrd 0x80200014 1]
set DMASR_BUSY 0x40000000

puts $outfile "DMA Status Register = $DMASR"

#while {1} {
for {set i 1} {$i<1000} {incr i} {
if {$DMASR != $DMASR_BUSY} {
puts $outfile [mwr 0x80200010 64]
}
}

puts $outfile [mrd 0x20000000  100]
puts $outfile [mrd 0x20002000  100]

close $outfile
exit
```

X1030_30_040908

Figure 30: **dma.tcl**

Figure 31 shows the Catalyst SPX4 Analyzer/Exerciser output after running the ep2rc_*.spf performance analyzer project.The payload throughput depends on various factors such as the size of the transfer, if print statements are included in the source code, and if the verification is included in the source code. For this run, all print statements are removed, there is no verification, and length is set to 20. The transfer is from XPS BRAM to Catalyst memory across the PCIe link. The data throughput is 19.0 MB/s and the payload throughput is 8.3 MB/s.

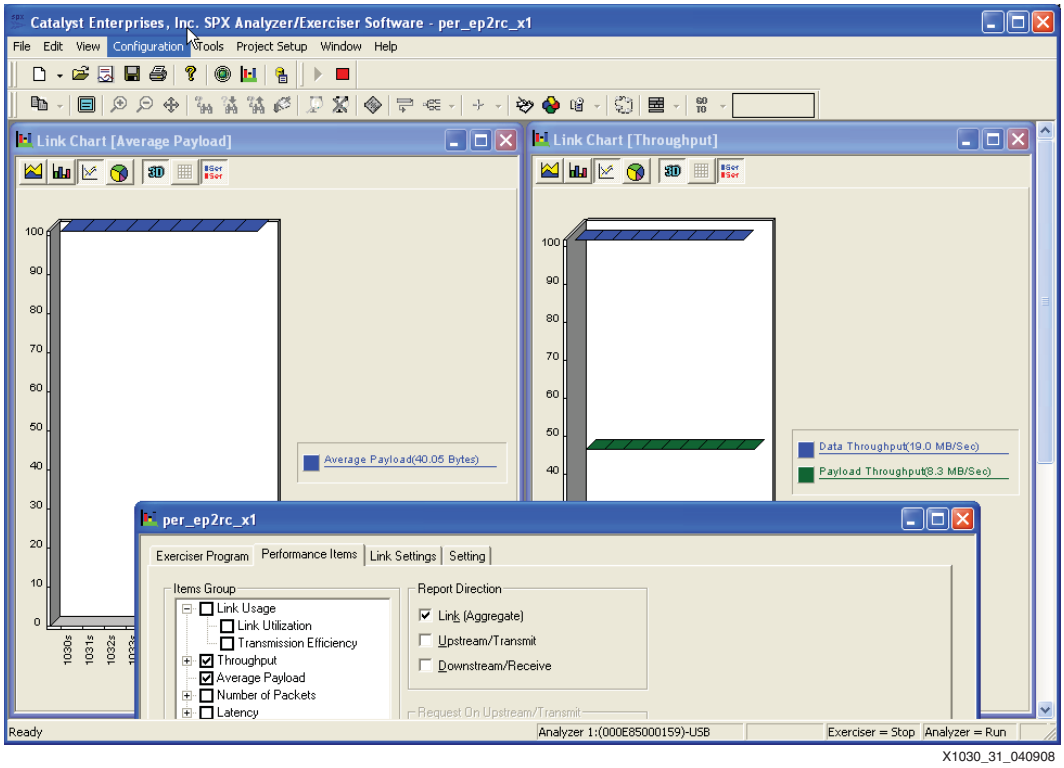


Figure 31: **EP to RC Performance Test Using XMD**

Figure 32 shows the performance of an Endpoint to Root Complex transaction using C code (pcie_dma_0.c) to generate stimuli with the length = 200. The data throughput is 61.8 MB/s and the payload throughput is 36.8 MB/s. In this test, the Source Address is XPS BRAM, which is 0x8AE10000, and the Destination Address is 0x20000000, which translates to Catalyst memory, across the PCIe link.

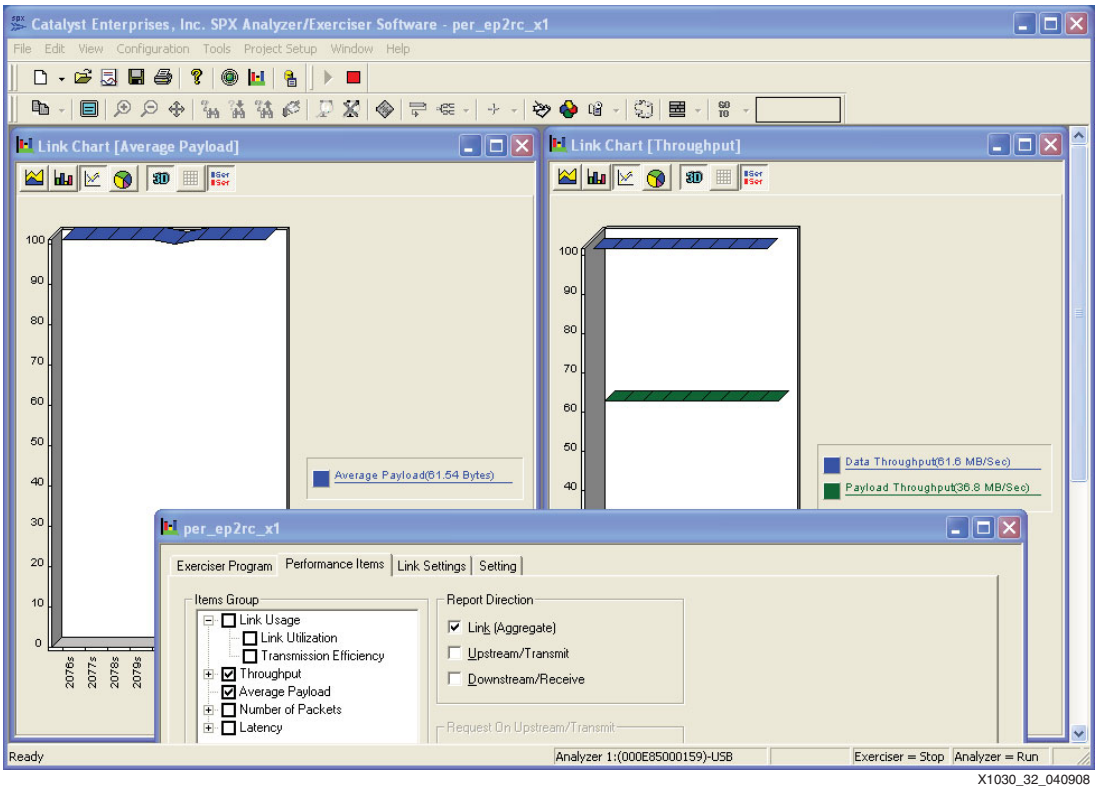


Figure 32: **EP to RC Performance Test Using C Code**

LeCroy Testing

Use the LeCroy - ML505 test setup shown in Figure 33 to verify the PLBv46 Endpoint Bridge using the LeCroy tester as root complex, including configuration and data transactions. The ML505 is inserted into the host emulator.

The `ml505_mb_plbv46_pcie/lecroy` directory contains the stimuli files which use `peg` as the filename extension.

This section discusses the procedures used in setting up the LeCroy, including defining the Recording and Generation Options. Root Complex to Endpoint transactions are discussed, followed by a section on Endpoint to Root Complex transactions.

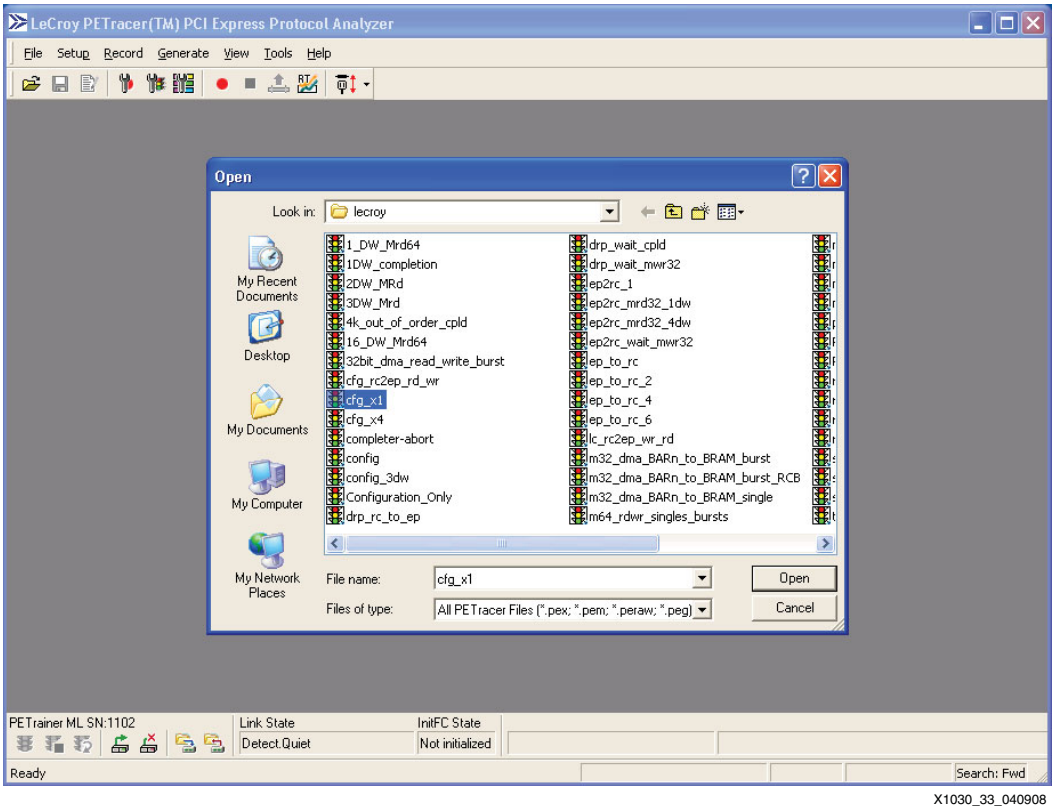


Figure 33: **LeCroy Test Setup**

Figure 34 is a photograph of the LeCroy test setup. The ML505 is inserted into the LeCroy Host Emulator. The Platform Cable USB Programming cable is connected to the ML505 JTAG connector.

X1030_34_040908

Figure 34: **LeCroy Test Equipment**

Figure 35 shows the menu for setting Generation Options after selecting Setup -> Generation Options.

The LeCroy ML test equipment is selected. Link Width is specified as x1. Select Host as the Interposer.

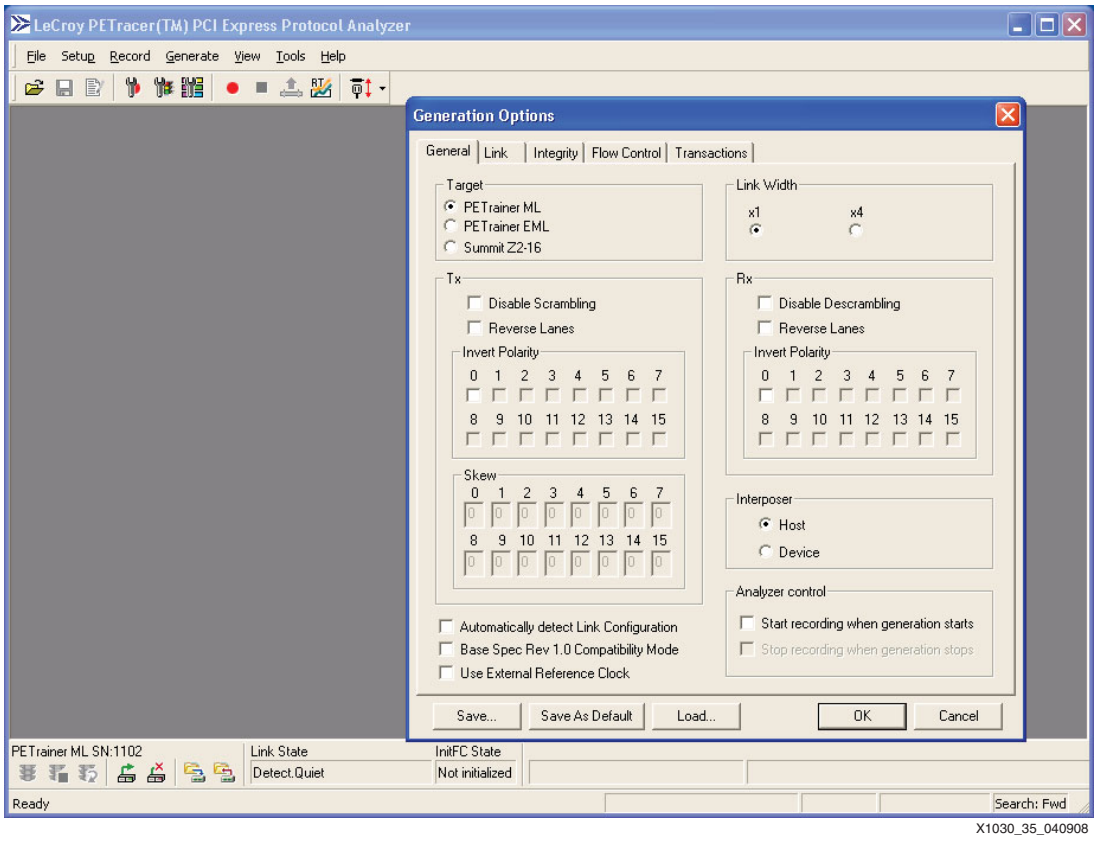


Figure 35: **Setting Generation Options**

Figure 36 shows the menu for setting Recording Options after selecting Setup -> Recording Options. The Simple Mode is used. An Event Trigger is selected.

The Buffer Size is specified as 32 MB and the Trigger Position is set at 90% post triggering.The x1 Lane Width is selected.

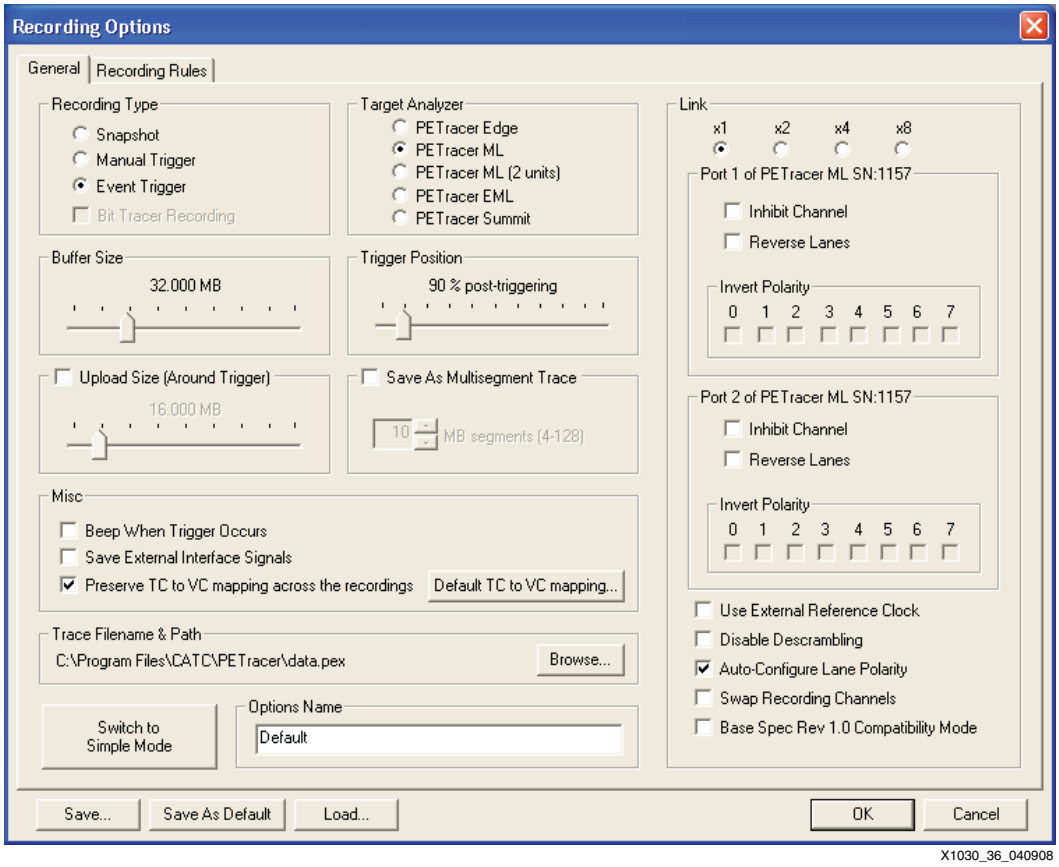


Figure 36: **Setting Recording Options**

Figure 37 shows using File -> Open to open a LeCroy stimuli (peg) file.

The LeCroy PETracer software provides the interface to the PETracer (Analyzer) and PETrainer (Exerciser). To run an analysis, click on the Record icon (the Sun) in the menu bar. Click the Traffic Light icon at the bottom left of the GUI. After the status bar indicates Traffic Finished, click the Stop icon (black filled square next to the Sun). This causes results to be shown in the Display area. Results files have a pex extension. Like peg files, pex files can be opened using File -> Open.

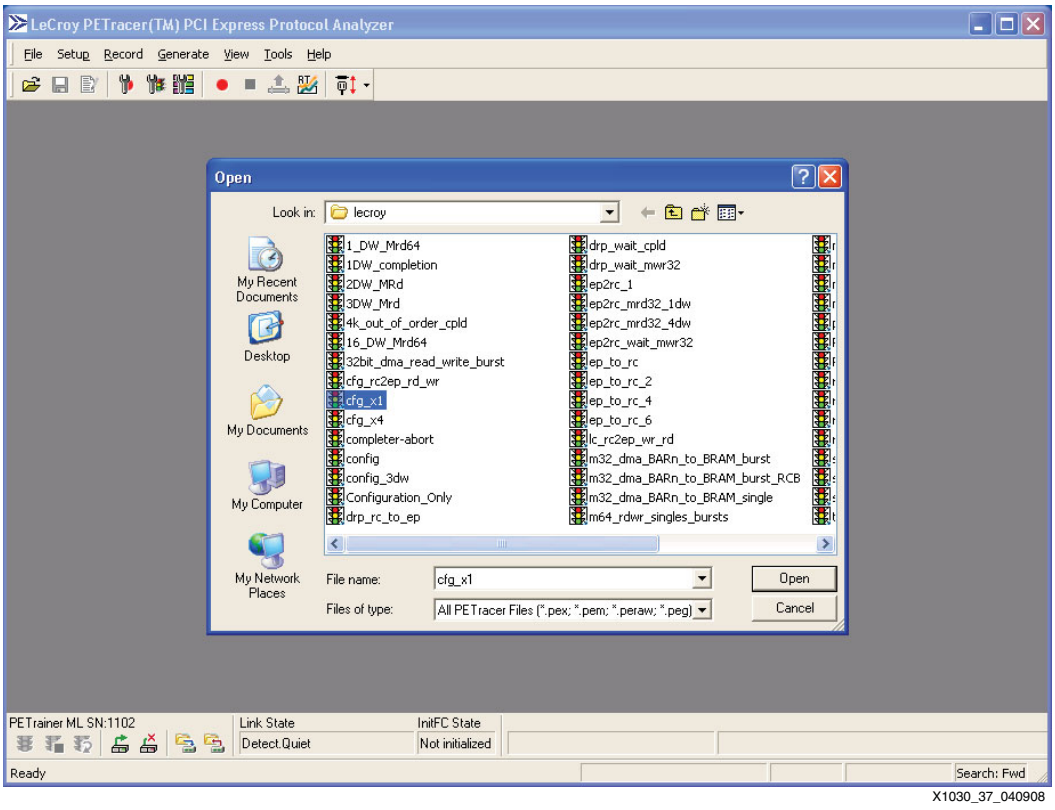


Figure 37: **Opening a LeCroy peg file**

Figure 38 shows the use of XMD to enable the Bridge Control Register. The BCR enables the Bus Master and the Base Address Registers (BARs).

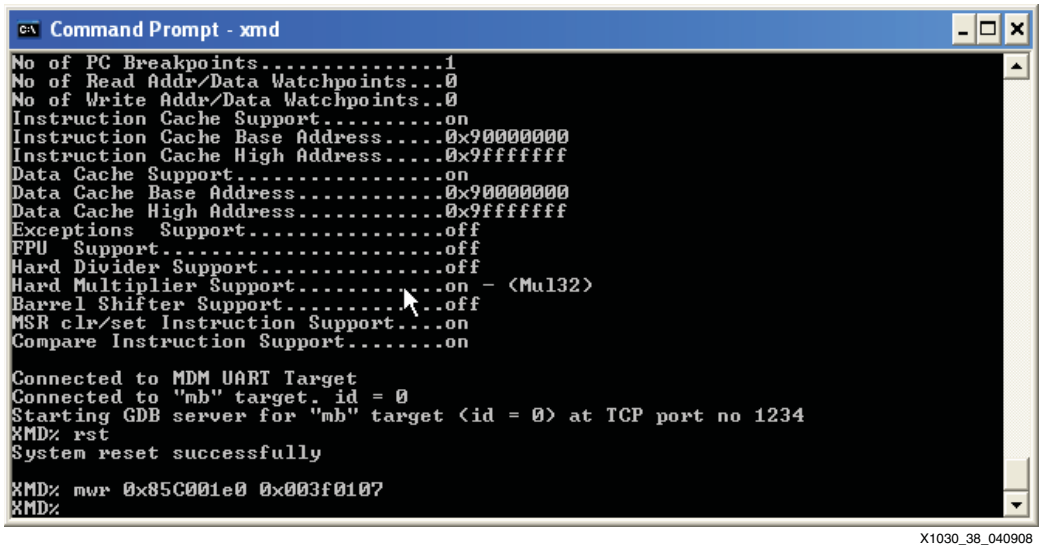

Command Prompt - xmd
No of PC Breakpoints.....1
No of Read Addr/Data Watchpoints...0
No of Write Addr/Data Watchpoints..0
Instruction Cache Support.....on
Instruction Cache Base Address.....0x90000000
Instruction Cache High Address.....0x9fffffff
Data Cache Support.....on
Data Cache Base Address.....0x90000000
Data Cache High Address.....0x9fffffff
Exceptions Support.....off
FPU Support.....off
Hard Divider Support.....off
Hard Multiplier Support.....on - (Mul32)
Barrel Shifter Support.....off
MSR clr/set Instruction Support....on
Compare Instruction Support.....on
Connected to MDM UART Target
Connected to "mb" target. id = 0
Starting GDB server for "mb" target (id = 0) at TCP port no 1234
XMD% rst
System reset successfully
XMD% mwr 0x85C001e0 0x003f0107
XMD%
X1030_38_040908

Figure 38: **Using XMD Commands to Write the Bridge Control Register**

After generation and recording options are specified and the BCR is written, the link must be trained. The Link State is displayed at the bottom of the PETracer GUI. Prior to training, the Link State is displayed as Detect.Quiet as shown at the bottom of Figure 37. After training, the Link State is displayed as L0. To initiate training, click on the Connect icon. To disable a trained link, click on the Disconnect icon.

Figure 39 shows LeCroy - ML505 PLBv46 Endpoint Bridge link is trained with the LTFSM in L0. If the clocking and resets are correct, link training occurs in less than one second. If link training is unsuccessful, the LTFSM cycles through training states.

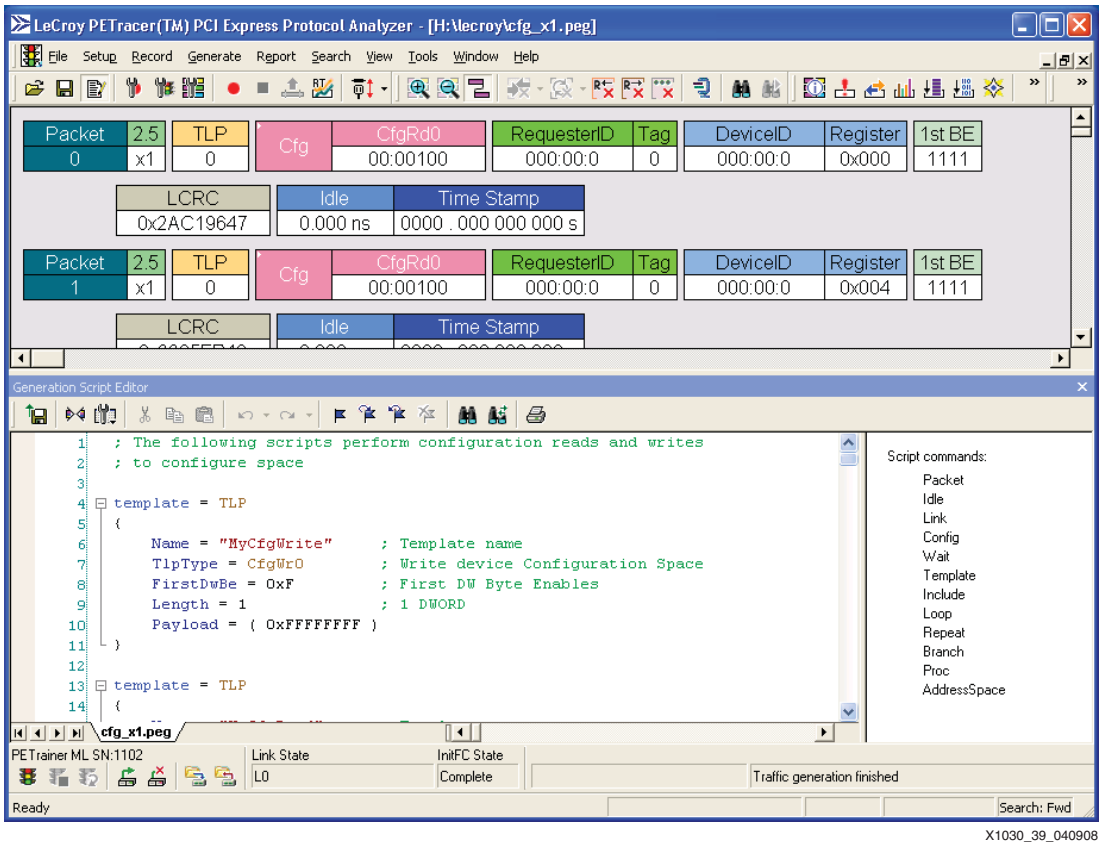

LeCroy PETracer(TM) PCI Express Protocol Analyzer - [H:\lecroy\cfg_x1.peg]
File Setup Record Generate Report Search View Tools Window Help
Packet 0 | 2.5 x1 | TLP 0 | Cfg | CfgRd0 00:00100 | RequesterID 000:00:0 | Tag 0 | DeviceID 000:00:0 | Register 0x000 | 1st BE 1111
LCRC 0x2AC19647 | Idle 0.000 ns | Time Stamp 0000 . 000 000 000 s
Packet 1 | 2.5 x1 | TLP 0 | Cfg | CfgRd0 00:00100 | RequesterID 000:00:0 | Tag 0 | DeviceID 000:00:0 | Register 0x004 | 1st BE 1111
LCRC | Idle | Time Stamp
Generation Script Editor
; The following scripts perform configuration reads and writes
; to configure space
template = TLP
{
Name = "MyCfgWrite" ; Template name
TlpType = CfgWr0 ; Write device Configuration Space
FirstDwBe = 0xF ; First DW Byte Enables
Length = 1 ; 1 DWORD
Payload = (0xFFFFFFFF)
}
template = TLP
{
Script commands:
Packet
Idle
Link
Config
Wait
Template
Include
Loop
Repeat
Branch
Proc
AddressSpace
cfg_x1.peg
PETrainer ML SN:1102
Link State
L0
InitFC State
Complete
Traffic generation finished
Ready
Search: Fwd
X1030_39_040908

Figure 39: **LeCroy After Link Trained**

Root Complex to Endpoint Transactions

As Root Complex, the LeCroy Trainer generates memory writes and memory reads to the ML505 memory. The ML505 reference design contains XPS BRAM and an MPMC interface to DDR2. Figure 40 shows the memory addressing for Root Complex (Catalyst) to Endpoint (ML505) transactions.The memory addressed is controlled by the BAR value written and by the C_PCIBAR2IPIFBAR_* generics. In the reference design, PCI BAR0 is written as 0x0000000060000000. C_PCIBAR2IPIFBAR_0 addresses XPS BRAM at 0x8AE10000, and C_PCIBAR2IPIFBAR_1 addresses DDR2 at location 0x90000000.

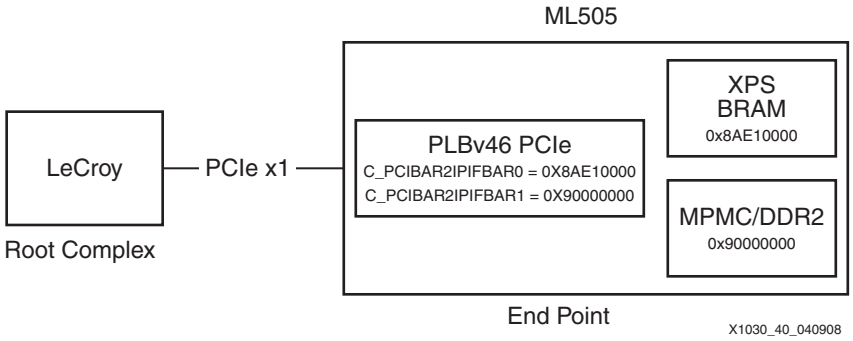
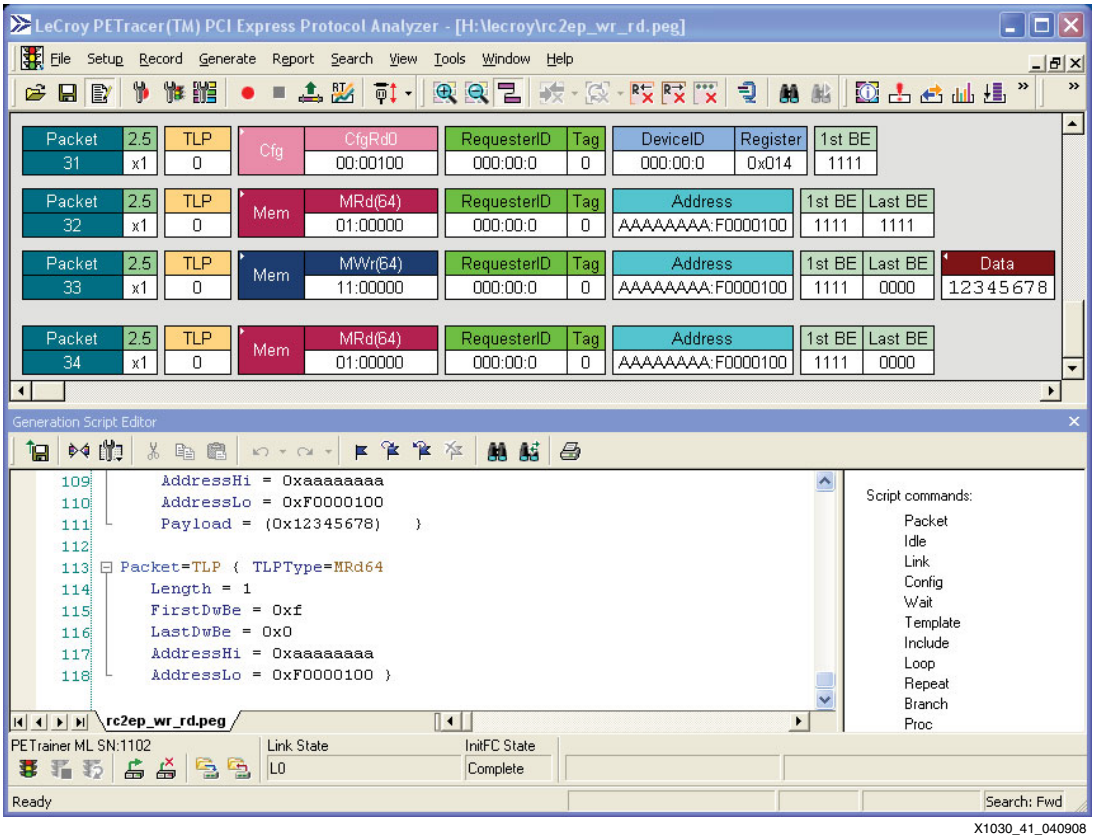


Figure 40: **LeCroy - ML505 Memory Addressing**

The display area shows the TLPs defined in the peg file. Figure 41 shows an excerpt from the rc2ep_wr_rd.peg file. The rc2ep_wr_rd.peg shown is writes FFFFFFFFs to the six BAR registers in the Configuration Space Header (CSH). This is done using the Repeat construct. The first register written is BAR0, located at offset 0x10. After writing and reading the CSH, packets 32 - 34 are MRd64, MW64 0x12345678, and MRd64


LeCroy PETracer(TM) PCI Express Protocol Analyzer - [H:\lecroy\rc2ep_wr_rd.peg]
File Setup Record Generate Report Search View Tools Window Help
Packet 31 2.5 x1 TLP 0 Cfg CfgRd0 00:00100 RequesterID 000:00:0 Tag 0 DeviceID 000:00:0 Register 0x014 1st BE 1111
Packet 32 2.5 x1 TLP 0 Mem MRd(64) 01:00000 RequesterID 000:00:0 Tag 0 Address AAAAAAAA:F0000100 1st BE 1111 Last BE 1111
Packet 33 2.5 x1 TLP 0 Mem MWr(64) 11:00000 RequesterID 000:00:0 Tag 0 Address AAAAAAAA:F0000100 1st BE 1111 Last BE 0000 Data 12345678
Packet 34 2.5 x1 TLP 0 Mem MRd(64) 01:00000 RequesterID 000:00:0 Tag 0 Address AAAAAAAA:F0000100 1st BE 1111 Last BE 0000
Generation Script Editor
109 AddressHi = 0xaaaaaaaa
110 AddressLo = 0xF0000100
111 Payload = (0x12345678) }
112
113 Packet=TLP { TLPType=MRd64
114 Length = 1
115 FirstDwBe = 0xf
116 LastDwBe = 0x0
117 AddressHi = 0xaaaaaaaa
118 AddressLo = 0xF0000100 }
Script commands:
Packet
Idle
Link
Config
Wait
Template
Include
Loop
Repeat
Branch
Proc
rc2ep_wr_rd.peg
PETrainer ML SN:1102
Link State L0
InitFC State Complete
Ready
Search: Fwd

X1030_41_040908

Figure 41: **RC to EP Write/Read Test**

The next figures show BAR0 configuration packets, followed by write, then read, operations on BAR0.

Figure 42 shows the configuration of BAR0 and the read, write, and read transactions. The address of BAR0 is 0x0000000060000000. Packet 0 is a CfgWr of the lower order address and packet 2 is a CfgWr of the higher order address. Packets 4 and 5 use CfgRd TLPs to verify the configuration writes. Packets 6, 7, and 8 are MRd32, MWr32, and MRd32 TLPs used to read and write BAR0 memory.

Double click on the Data field in packet 7 to display the 1234678 value.

The endianess of the address in the CfgWr0 TLP differs from the endianess of the address in the MWr32 and MRd32 TLPs.

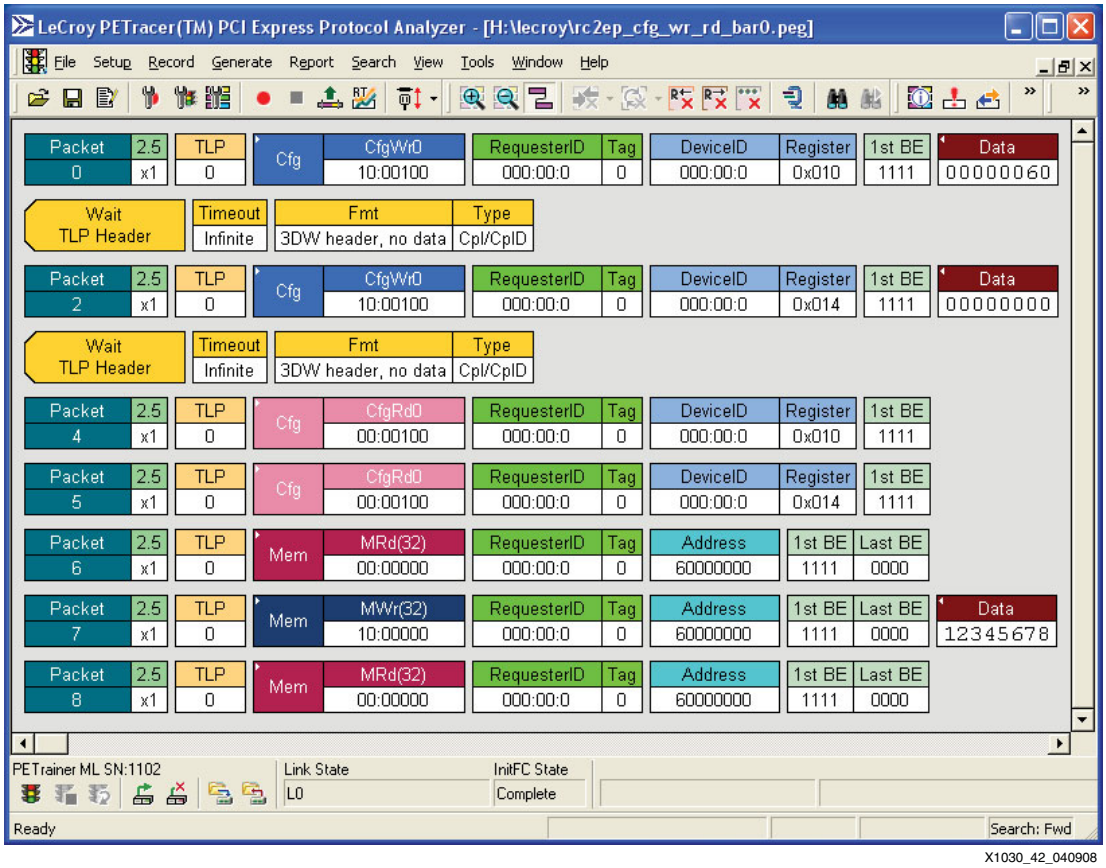

LeCroy PETracer(TM) PCI Express Protocol Analyzer - [H:\lecroy\rc2ep_cfg_wr_rd_bar0.peg]
File Setup Record Generate Report Search View Tools Window Help
Packet 0 | 2.5 x1 | TLP 0 | Cfg | CfgWr0 10:00100 | RequesterID 000:00:0 | Tag 0 | DeviceID 000:00:0 | Register 0x010 | 1st BE 1111 | Data 00000060
Wait TLP Header | Timeout Infinite | Fmt 3DW header, no data | Type Cpl/CplD
Packet 2 | 2.5 x1 | TLP 0 | Cfg | CfgWr0 10:00100 | RequesterID 000:00:0 | Tag 0 | DeviceID 000:00:0 | Register 0x014 | 1st BE 1111 | Data 00000000
Wait TLP Header | Timeout Infinite | Fmt 3DW header, no data | Type Cpl/CplD
Packet 4 | 2.5 x1 | TLP 0 | Cfg | CfgRd0 00:00100 | RequesterID 000:00:0 | Tag 0 | DeviceID 000:00:0 | Register 0x010 | 1st BE 1111
Packet 5 | 2.5 x1 | TLP 0 | Cfg | CfgRd0 00:00100 | RequesterID 000:00:0 | Tag 0 | DeviceID 000:00:0 | Register 0x014 | 1st BE 1111
Packet 6 | 2.5 x1 | TLP 0 | Mem | MRd(32) 00:00000 | RequesterID 000:00:0 | Tag 0 | Address 60000000 | 1st BE 1111 | Last BE 0000
Packet 7 | 2.5 x1 | TLP 0 | Mem | MWr(32) 10:00000 | RequesterID 000:00:0 | Tag 0 | Address 60000000 | 1st BE 1111 | Last BE 0000 | Data 12345678
Packet 8 | 2.5 x1 | TLP 0 | Mem | MRd(32) 00:00000 | RequesterID 000:00:0 | Tag 0 | Address 60000000 | 1st BE 1111 | Last BE 0000
PETrainer ML SN:1102
Link State L0
InitFC State Complete
Ready
Search: Fwd
X1030_42_040908

Figure 42: **Configuring and Testing BAR0**

Figure 43 shows the results after running rc2ep_cfg_wr_rd_bar0.peg. Packet 9 is a MWr32 of 0x12345678 to address 0x0000000060000000. The address is translated using the generic C_PCIBAR2IPIFBAR_0 to XPS BRAM at 0x8AE10000. In packet 12, the data value 0x12345678 is returned in the CplD packet.

The status fields indicate Successful Completion (SC).

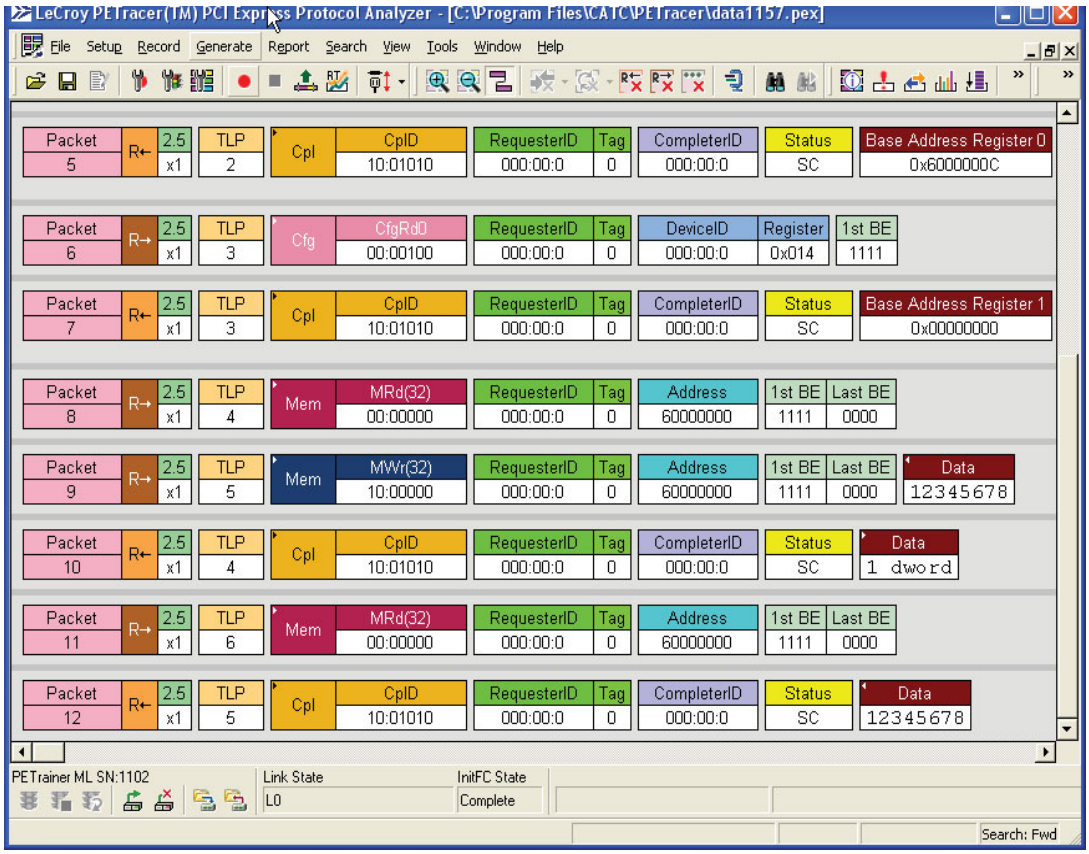

LeCroy PETracer(TM) PCI Express Protocol Analyzer - [C:\Program Files\CATC\PETracer\data1157.pex]
File Setup Record Generate Report Search View Tools Window Help
Packet 5 R← 2.5 x1 TLP 2 Cpl CplD 10:01010 RequesterID 000:00:0 Tag 0 CompleterID 000:00:0 Status SC Base Address Register 0 0x6000000C
Packet 6 R→ 2.5 x1 TLP 3 Cfg CfgRd0 00:00100 RequesterID 000:00:0 Tag 0 DeviceID 000:00:0 Register 0x014 1st BE 1111
Packet 7 R← 2.5 x1 TLP 3 Cpl CplD 10:01010 RequesterID 000:00:0 Tag 0 CompleterID 000:00:0 Status SC Base Address Register 1 0x00000000
Packet 8 R→ 2.5 x1 TLP 4 Mem MRd(32) 00:00000 RequesterID 000:00:0 Tag 0 Address 60000000 1st BE 1111 Last BE 0000
Packet 9 R→ 2.5 x1 TLP 5 Mem MWr(32) 10:00000 RequesterID 000:00:0 Tag 0 Address 60000000 1st BE 1111 Last BE 0000 Data 12345678
Packet 10 R← 2.5 x1 TLP 4 Cpl CplD 10:01010 RequesterID 000:00:0 Tag 0 CompleterID 000:00:0 Status SC Data 1 dword
Packet 11 R→ 2.5 x1 TLP 6 Mem MRd(32) 00:00000 RequesterID 000:00:0 Tag 0 Address 60000000 1st BE 1111 Last BE 0000
Packet 12 R← 2.5 x1 TLP 5 Cpl CplD 10:01010 RequesterID 000:00:0 Tag 0 CompleterID 000:00:0 Status SC Data 12345678
PETrainer ML SN:1102
Link State L0
InitFC State Complete
Search: Fwd

Figure 43: **BAR0 Test Results**

Figure 44 shows the verification of the Endpoint to Root Complex PCIe transactions using XMD. In the system.mhs, the PLBv46 Endpoint Bridge generic C_PCIBAR2IPIFBAR0 is 0x8AE10000, the location of XPS BRAM. This shows that the 0x12346578 written by the LeCroy Root Complex MWr64 TLP is resident in XPS BRAM.

Figure 44: **XMD Verification of BAR0 Tests**

Figure 45 shows an excerpt of a peg file.

The peg file used as stimuli in LeCroy transactions is readable and editable. In the figure, templates are defined for Configuration Write and Configuration Read TLPs. The Configuration Write template is called in the repeat loop to write FFFFFFFFs to the six Configuration Space Header BARs.

The peg files in `ml505_mb_plbv46_pcie/lecroy` can be used to test the PLBv46 Endpoint Bridge on the ML505.

```
template = TLP{
Name = "MyCfgWrite"
; Template name
TlpType = CfgWr0
; Write device Configuration Space
FirstDwBe = 0xF;
 First DW Byte Enables
Length = 1
;1 DWORD
Payload = ( 0xFFFFFFFF )
template = TLP
{
Name = "MyCfgRead"
; Template name
TlpType = CfgRd0
; Read device Configuration Space
FirstDwBe = 0xF; First DW Byte Enables
Length = 1
; 1 DWORD
}
; Enumerate all 6 Base Address registers
repeat = Begin { Count = 6 Counter = i }
; Write 0xFFFFFFFF into Base Address register
packet = "MyCfgWrite" {
Register = ( 0x10 + i * 4 )
; Wait for completion received
wait = TLP {
      TLPType = Cpl
}
; Read Base Address register
packet = "MyCfgRead" {
      Register = ( 0x10 + i * 4 )
}
; Wait for completion received
wait = TLP {
      TLPType = CplD
}

repeat = End
```

X1030_45_040908

Figure 45: **peg Example**

Endpoint to Root Complex Transactions

In Endpoint to Root Complex transactions, the read and write operations originate from the ML505 and target the LeCroy. The LeCroy model used in this application note, the PCI Express Multi-Lane (ML) Exerciser/Analyzer, does not have target memory. For read operations, the peg files are written to respond with read data.

Invoke PETracer and run **File** → **Open lecroy/ep2rc_mrd32_1dw**.

Endpoint to Root Complex transactions are generated with XMD commands or C code. Since the MWr and MRd TLPs originate from the ML505, the LeCroy peg files cause the LeCroy to wait for the TLP(s) from the ML505. Figure 46 shows the peg for the EP to RC MRd32. The LeCroy waits for the MRd32 packet from the ML505. When the MRd32 packet is received, the LeCroy returns a Completion with Data (CplD) packet with a 0x12345678 payload.

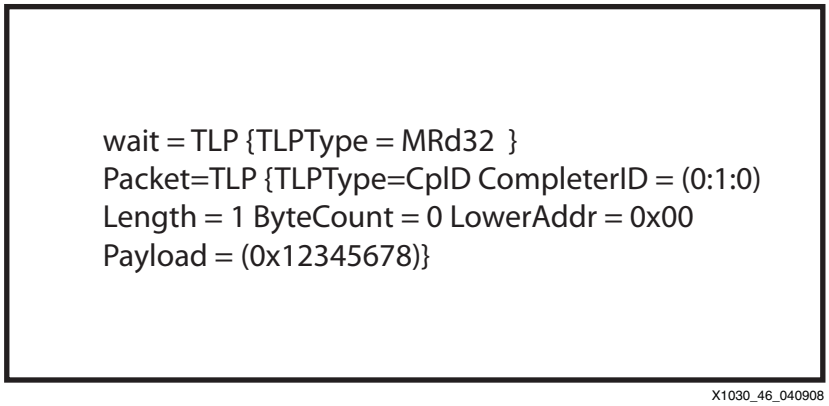

wait = TLP {TLPType = MRd32 }
Packet=TLP {TLPType=CplD CompleterID = (0:1:0)
Length = 1 ByteCount = 0 LowerAddr = 0x00
Payload = (0x12345678)}
X1030_46_040908

Figure 46: **ep2rc_mrd32**

Figure 47 defines the functionality of the LeCroy Root Complex when receiving a MRd32 transaction from the PLBv46 Endpoint Bridge endpoint on the ML505.

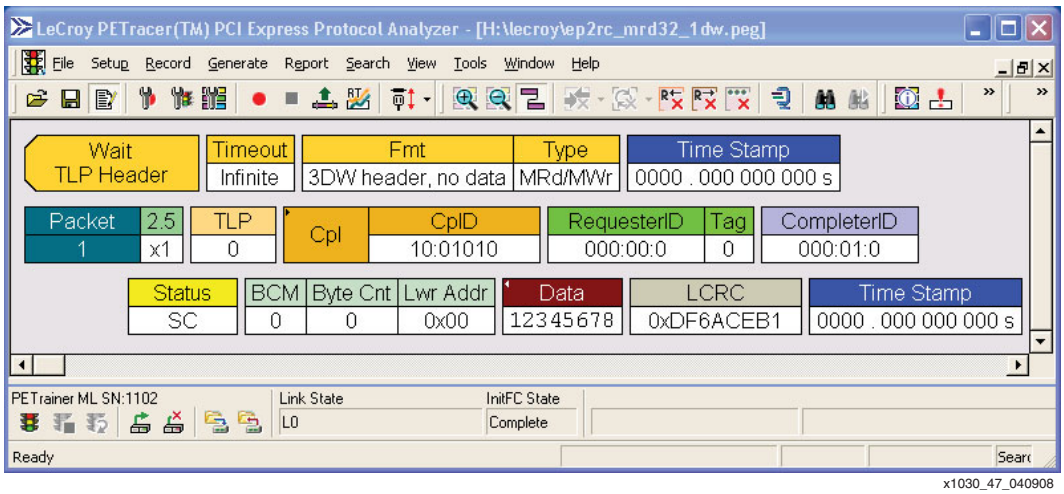
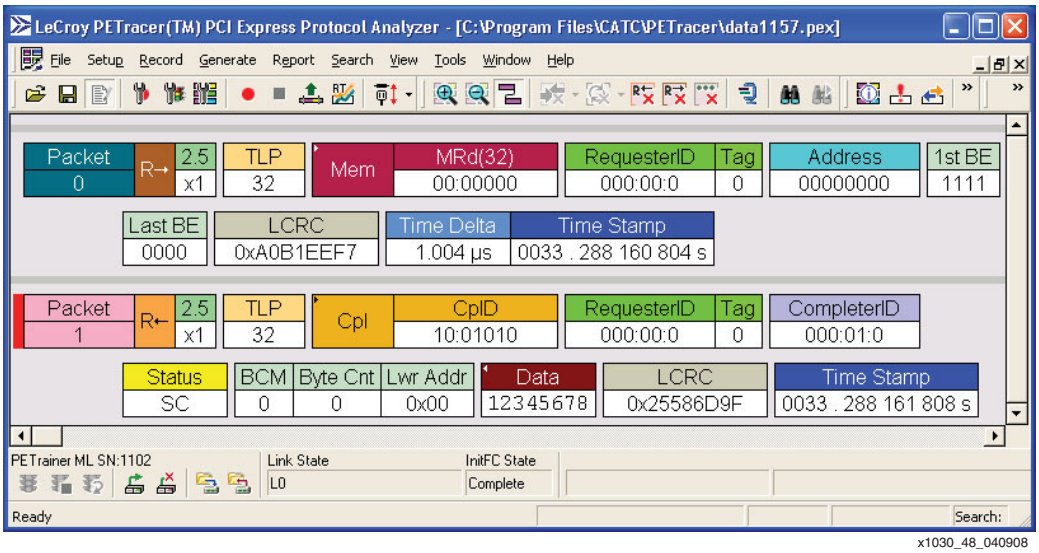


Figure 47: **EP to RC MRd32 Test Stimuli (1 DW)**

Figure 48 shows results from running the EP to RC memory read. The peg is loaded. Start recording by clicking on the Sun icon in the menu bar. Click the Traffic Light icon. Generate a 1 read using XMD.

```
mrd 0x20000000 1
```

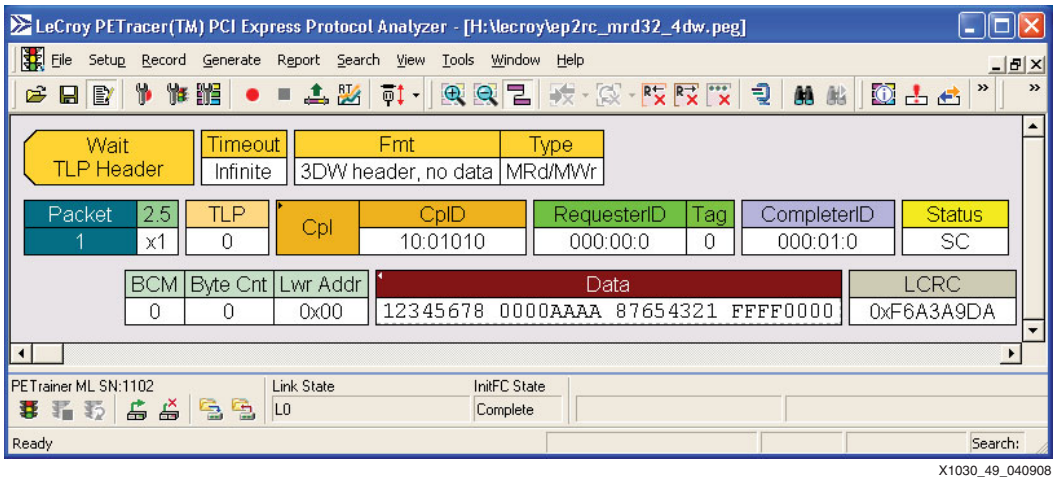
Click the Black Square to stop recording and view the results.



x1030_48_040908

Figure 48: **EP to RC MRd32 Test Results (1 DW)**

Figure 49 shows the ep2rc_mrd32_4dw.peg for a four doubleword Endpoint to Root Complex MRd32.

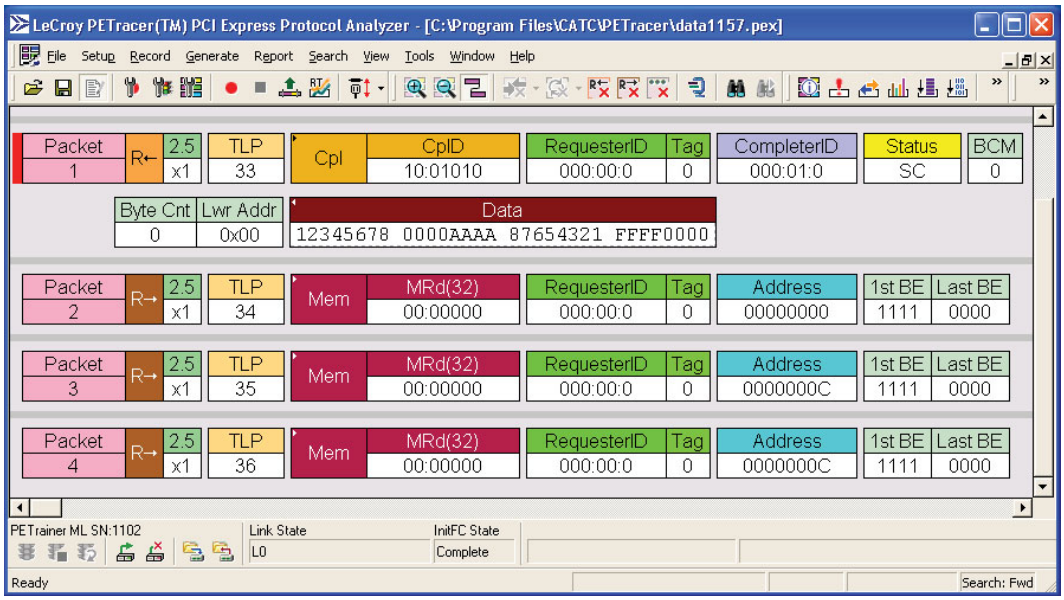


X1030_49_040908

Figure 49: **EP to RC MRd32 Test Stimuli (4 DW)**

Figure 50 shows results from running the XMD command below.

```
mrd 0x20000000 4
```


LeCroy PETracer(TM) PCI Express Protocol Analyzer - [C:\Program Files\CATC\PETracer\data1157.pex]
File Setup Record Generate Report Search View Tools Window Help
Packet 1 R← 2.5 x1 TLP 33 Cpl CplD 10:01010 RequesterID 000:00:0 Tag 0 CompleterID 000:01:0 Status SC BCM 0
Byte Cnt 0 Lwr Addr 0x00 Data 12345678 0000AAAA 87654321 FFFF0000
Packet 2 R→ 2.5 x1 TLP 34 Mem MRd(32) 00:00000 RequesterID 000:00:0 Tag 0 Address 00000000 1st BE 1111 Last BE 0000
Packet 3 R→ 2.5 x1 TLP 35 Mem MRd(32) 00:00000 RequesterID 000:00:0 Tag 0 Address 0000000C 1st BE 1111 Last BE 0000
Packet 4 R→ 2.5 x1 TLP 36 Mem MRd(32) 00:00000 RequesterID 000:00:0 Tag 0 Address 0000000C 1st BE 1111 Last BE 0000
PETrainer ML SN:1102
Link State L0
InitFC State Complete
Ready
Search: Fwd

X1030_50_040908

Figure 50: **EP to RC MRd32 Test Results (4 DW)**

Endpoint to Root Complex Write Transactions

Figure 51 shows the peg for the EP to RC MWr32. As with EP to RC memory reads, start recording by clicking on the Sun icon, and then click on the traffic light.

```
wait = TLP {
TLPType = MWr32
 }
```

X1030_51_040908

Figure 51: **ep2rc_wait_mwr32.peg**

Figure 52 shows LeCroy Root Complex setup for analyzing an Endpoint to Root Complex MWr32 operation.

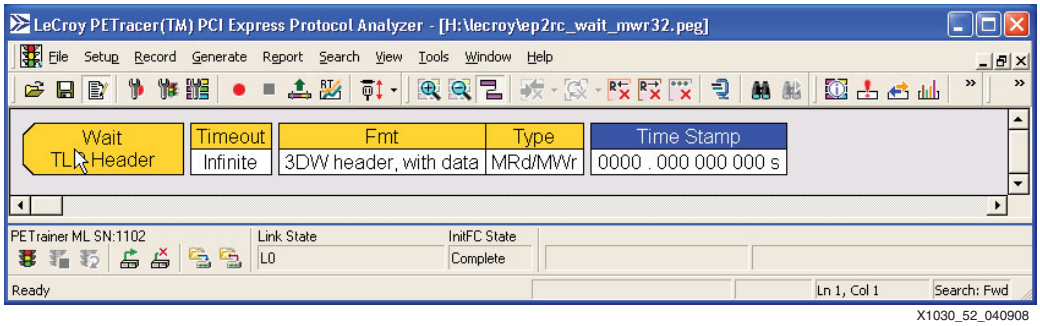


Figure 52: **EP to RC - Write Operation**

The xmd command below generates the stimuli for the PLBv46 Endpoint Bridge to transmit the TLP.

```
mwr 0x20000000 0x12345678
```

Figure 53 shows the Analyzer output for an EP to Root Complex Memory Write of 0x12345678.

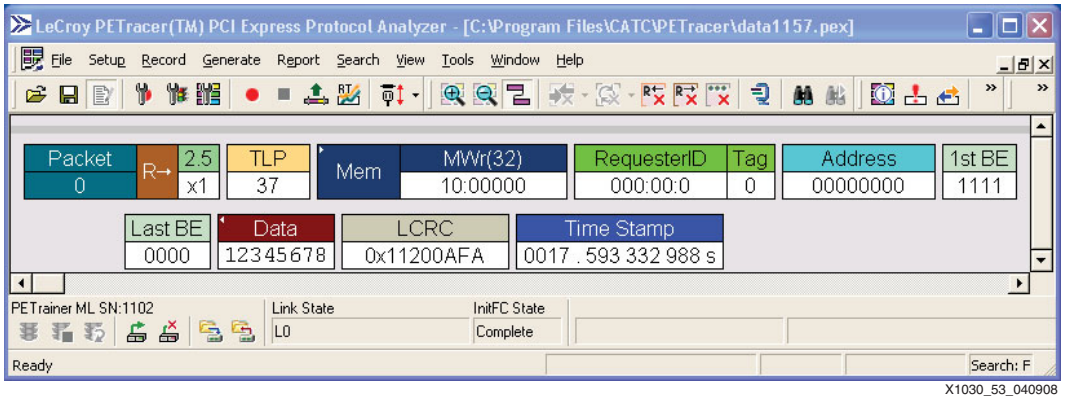
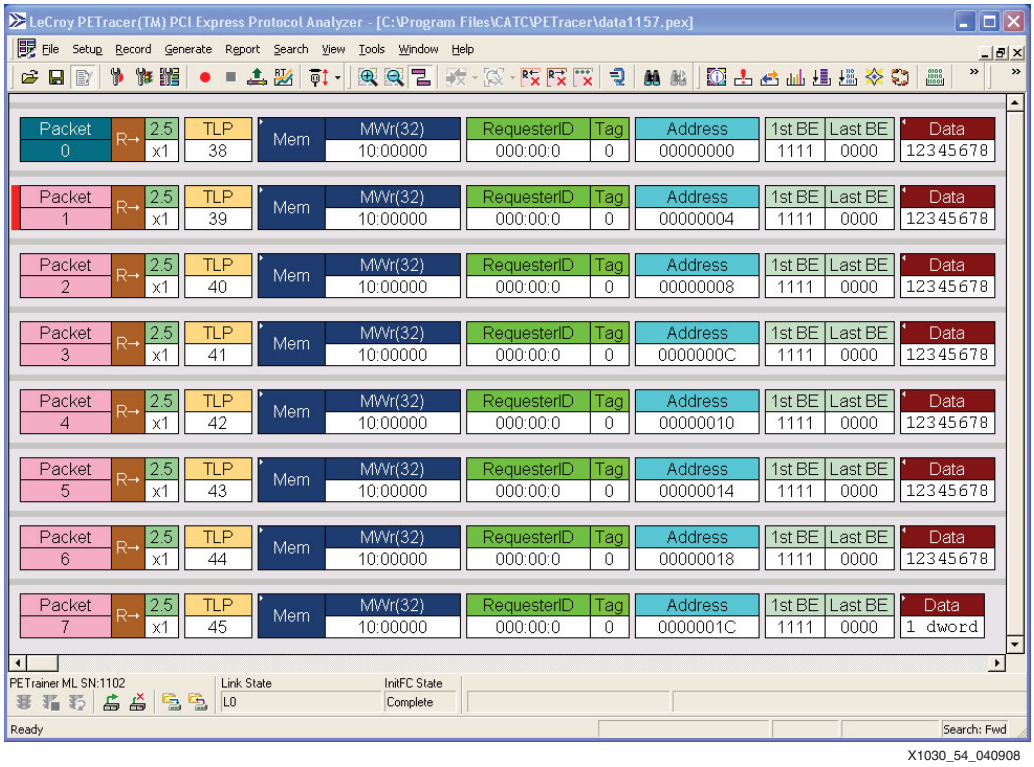


Figure 53: **EP to RC Write Results**

The write operation is easily varied using XMD. The XMD command below writes eight locations.

mwr 0x20000000 0x12345678 8

Figure 54 shows the results from running the eight doubleword Endpoint to Root Complex write transaction.


LeCroy PETracer(TM) PCI Express Protocol Analyzer - [C:\Program Files\CATC\PETracer\data1157.pex]
File Setup Record Generate Report Search View Tools Window Help
Packet 0 R→ 2.5 x1 TLP 38 Mem MWr(32) 10:00000 RequesterID 000:00:0 Tag 0 Address 00000000 1st BE 1111 Last BE 0000 Data 12345678
Packet 1 R→ 2.5 x1 TLP 39 Mem MWr(32) 10:00000 RequesterID 000:00:0 Tag 0 Address 00000004 1st BE 1111 Last BE 0000 Data 12345678
Packet 2 R→ 2.5 x1 TLP 40 Mem MWr(32) 10:00000 RequesterID 000:00:0 Tag 0 Address 00000008 1st BE 1111 Last BE 0000 Data 12345678
Packet 3 R→ 2.5 x1 TLP 41 Mem MWr(32) 10:00000 RequesterID 000:00:0 Tag 0 Address 0000000C 1st BE 1111 Last BE 0000 Data 12345678
Packet 4 R→ 2.5 x1 TLP 42 Mem MWr(32) 10:00000 RequesterID 000:00:0 Tag 0 Address 00000010 1st BE 1111 Last BE 0000 Data 12345678
Packet 5 R→ 2.5 x1 TLP 43 Mem MWr(32) 10:00000 RequesterID 000:00:0 Tag 0 Address 00000014 1st BE 1111 Last BE 0000 Data 12345678
Packet 6 R→ 2.5 x1 TLP 44 Mem MWr(32) 10:00000 RequesterID 000:00:0 Tag 0 Address 00000018 1st BE 1111 Last BE 0000 Data 12345678
Packet 7 R→ 2.5 x1 TLP 45 Mem MWr(32) 10:00000 RequesterID 000:00:0 Tag 0 Address 0000001C 1st BE 1111 Last BE 0000 Data 1 dword
PETrainer ML SN:1102
Link State L0
InitFC State Complete
Ready
Search: Fwd

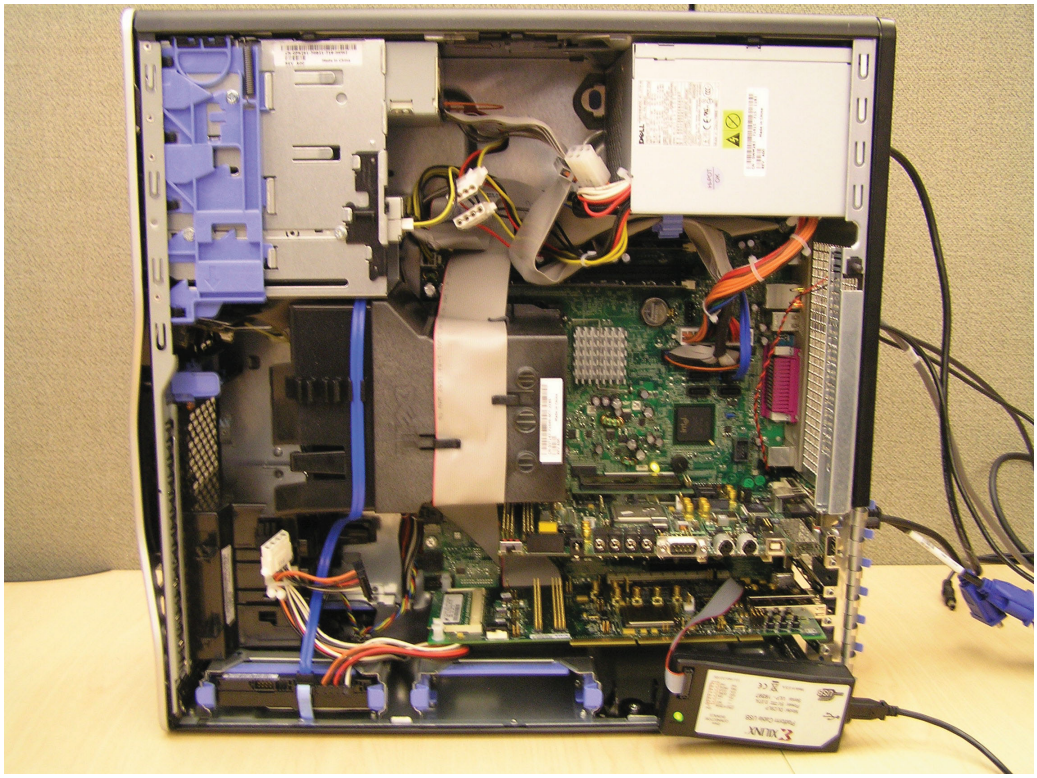
X1030_54_040908

Figure 54: **EP to RC Write Results - 8 DW**

Testing with a PC

Using a Personal Computer (PC) as Root Complex is an inexpensive method of verifying PLBv46 Endpoint Bridge functionality. PCItree and the Memory Endpoint Test run on PCs. Figure 55 shows the ML505 in a Dell 390 PC. The PC runs Windows XP and has the ISE, EDK, and PCItree software installed. The PC PCIe integrated circuits act as root complex. The Dell 390 has a x1 connector for PCIe slot 1 and a x8 connector for PCIe slot 4. In the Dell 390, only 4 of the 8 lanes of the x8 connector are active. The ML505 is powered from the ML505 power supply. The ML505 is inserted in PCIe slot 1.

The USB Platform Cable is connected to the ML505 JTAG port for Impact, XMD, and ChipScope operations. A Serial Communication Cable is connected to communicate to a communication terminal.



X1030_55_040908

Figure 55: **PC Test Environment**

The power up sequence of the PC affects the PCIe scan. In order for BIOS to recognize the drivers and PCIe BARs at power up, the FPGA bit file should be loaded prior to PC power up.

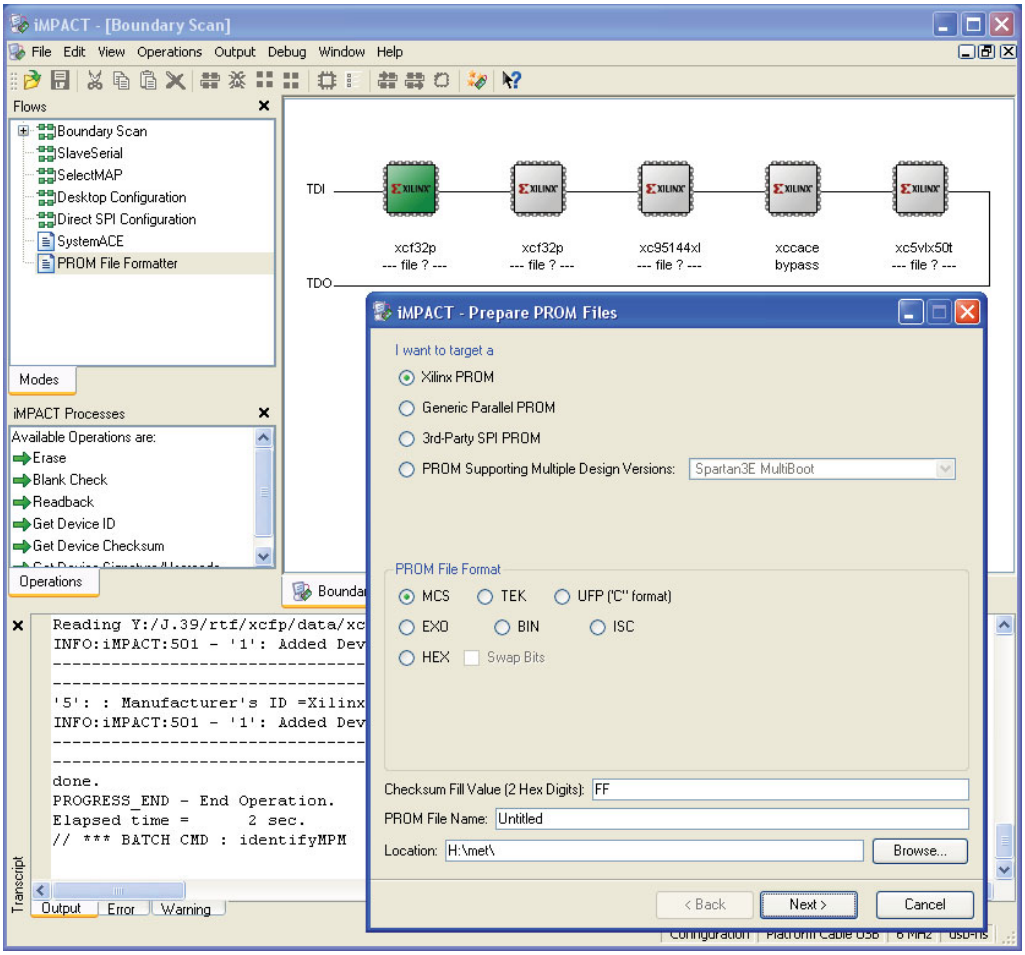
It is possible to configure the FPGA after PC power up using JTAG mode, but a warm restart is usually required to get a PCI scan to work. A warm restart is a PC Shutdown with Restart. Xilinx recommends writing the ML505 XCF32P PROM with the contents of the MCS file so that configuration occurs at power up.

Xilinx recommends the use of the Master SelectMap configuration mode. Since it is faster than Master Serial mode, the ML505 is more likely to be configured at the time of the PCIe scan.

Configuring the ML505 vlx50t when used in a PC PCIe Slot

The `ml505_mb_plbv46_pcie/ready_for_download/ml505_mb_plbv46_pcie.mcs` is the configuration file for this reference design. Because in PC based application it can save so much time to configure from the PROM, the next figures provide the steps for creating a mcs for the ML505.

Figure 56 shows the ML505 Boundary Scan chain. The first XCF32P is used to configure the FPGA. Right clicking on the XCF32P invokes the Prepare PROM GUI.



X1030_56_040908

Figure 56: **ML505 Boundary Scan Chain**

Provide the PROM file name as shown in Figure 57.

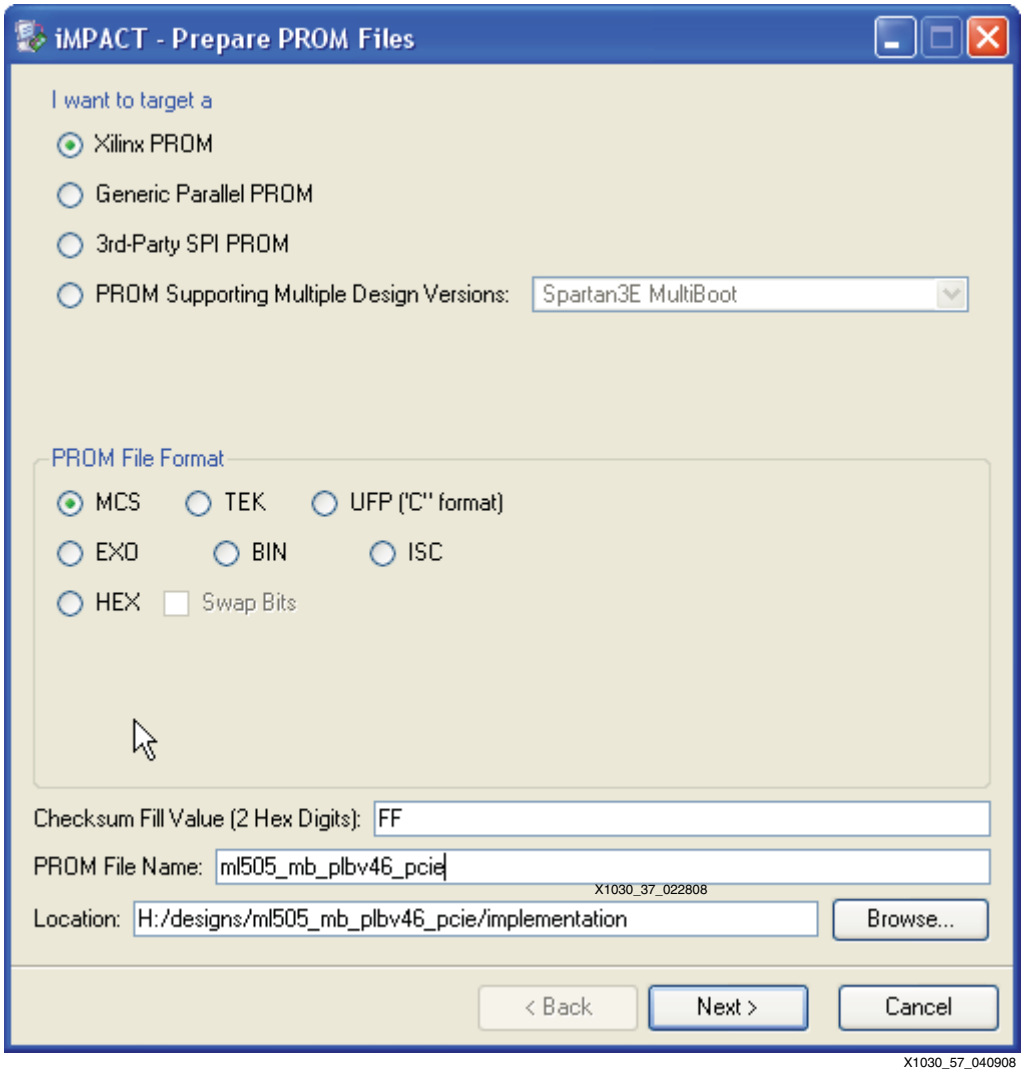


Figure 57: **Defining the PROM File**

Specify the XCF32P PROM as shown in Figure 58.

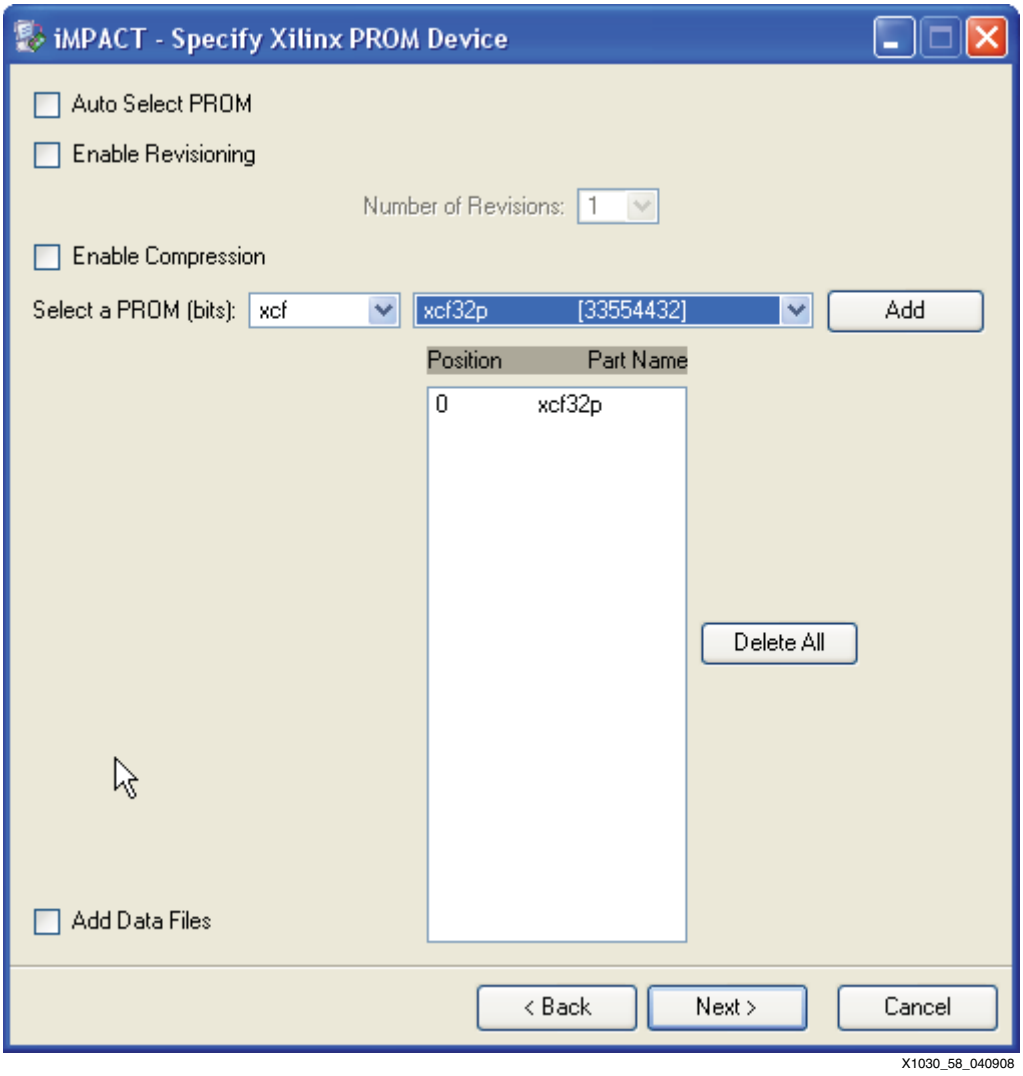


Figure 58: **Specifying the XCF32P PROM**

Select the bit file (download.bit) as shown in Figure 59.

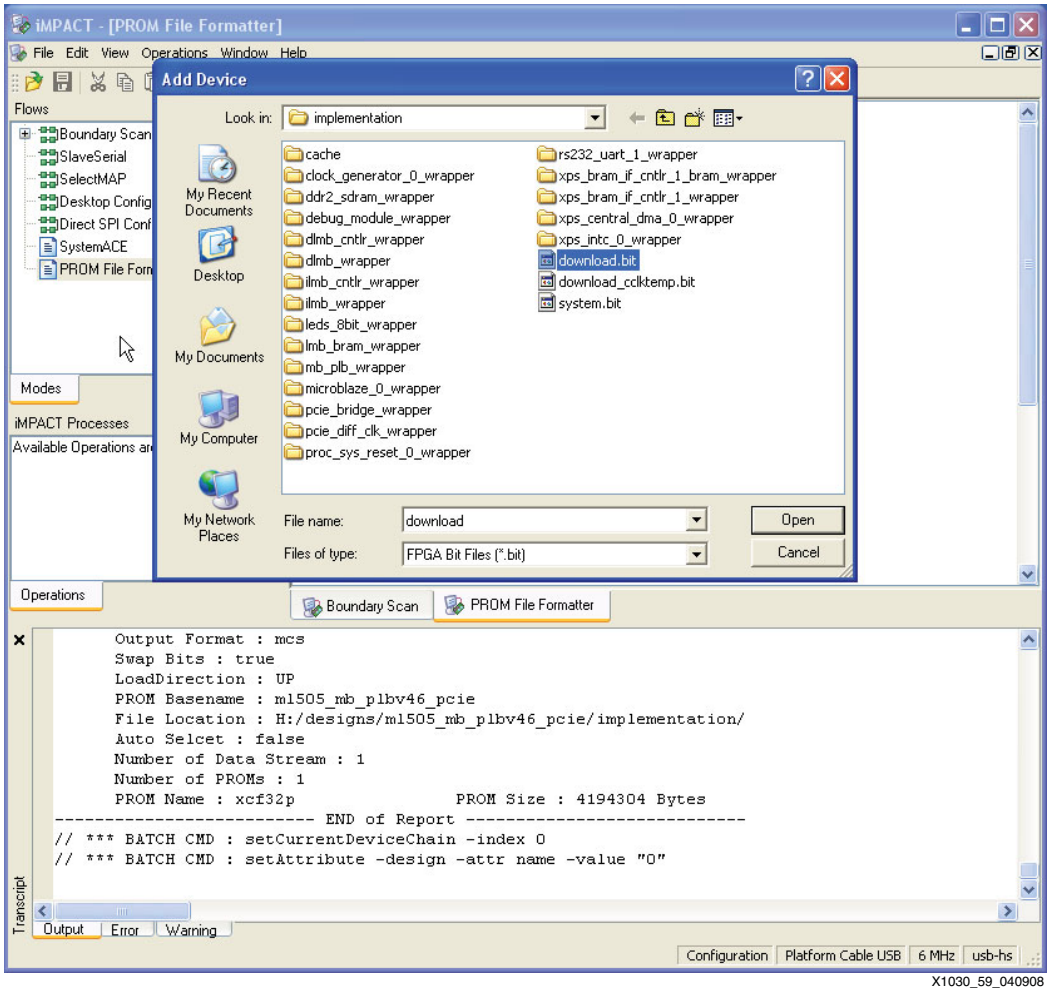


Figure 59: **Specifying the Bit File**

Figure 60 shows the generated mcs file.

Users generating th ePROM file for the first time should reference pages 101-107 of UG201 (v1.4) Virtex-5 FPGA ML555 Development Kit for PCI and PCI Express Designs.

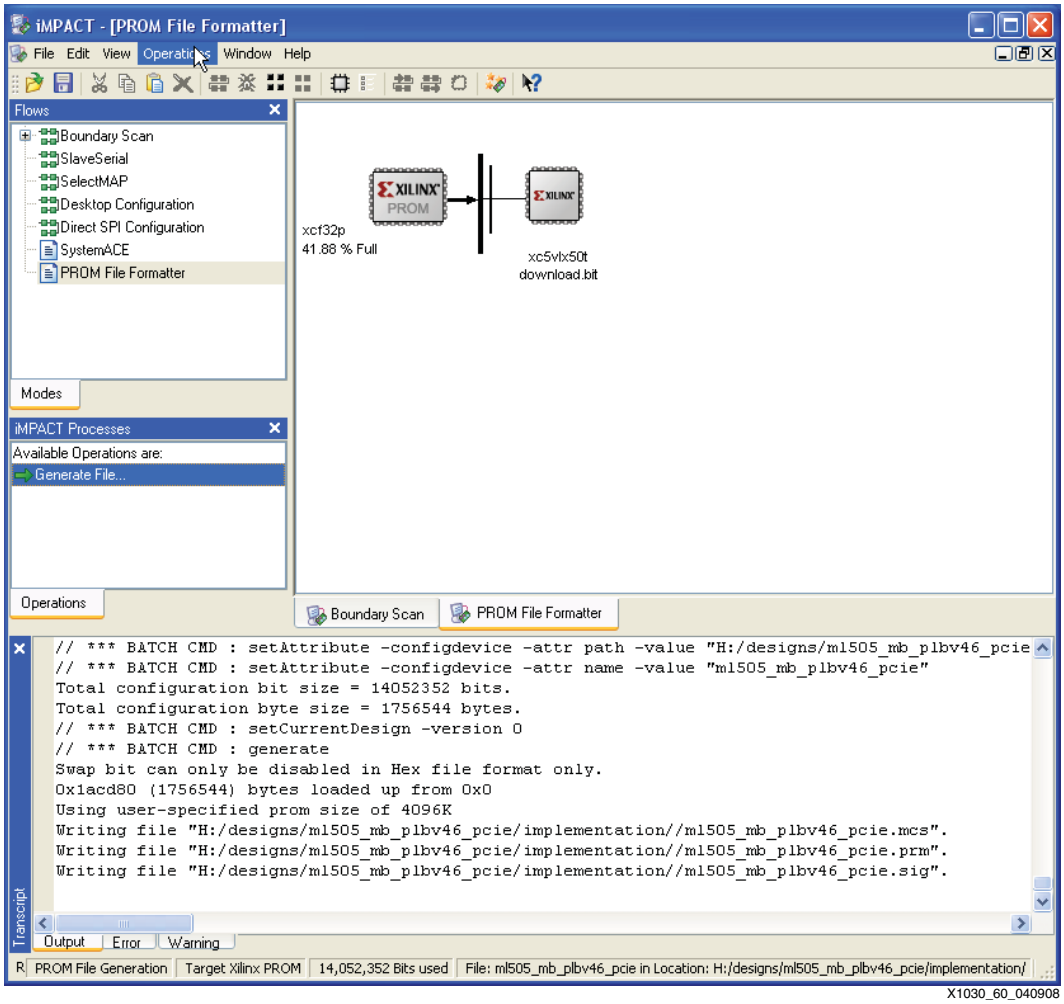


Figure 60: **Selecting Generate File**

The recommended configuration mode is Master SelectMap, which is specified when the configuration Mode Switch (SW3) should be set to M0-0 (ON), M1-0, M2-1.

Use Impact to download the mcs file into the ML505 XCF32 PROM. Select the XCF32P, left click to invoke a menu, and select Program. Under the **Programming Properties** menu, check **Parallel Mode** under **PROM Specific Properties**.

Insert the ML505 into the PCIe slot and power-on the PC. Verify that the DONE LED is lit.

PCItree Testing

PCItree is shareware available from http://www.pcitree.de. It runs on Windows XP. PCItree can be used for either PCI or PCIe tests. In the tests described in this section, the ML505 Embedded Development Platform is inserted into a Dell 390 x1 slot for the ml505_mb_plbv46_pcie project.

Invoke XMD and enable the master and BARs by writing to the Bridge Control Register.

```
mwr 0x85C001E0 0x003F0107
```

Figure 61 shows the XMD output when the PLBv46 Endpoint Bridge configuration space header registers are read. At power-up, the Device ID is 0x0505 and the Vendor ID is 0x10EE. BAR0 is 0x0000000C. The values are displayed in Big Endian format (EE100505). The address value at offset 10 contains the addressing size and prefetchability fields.

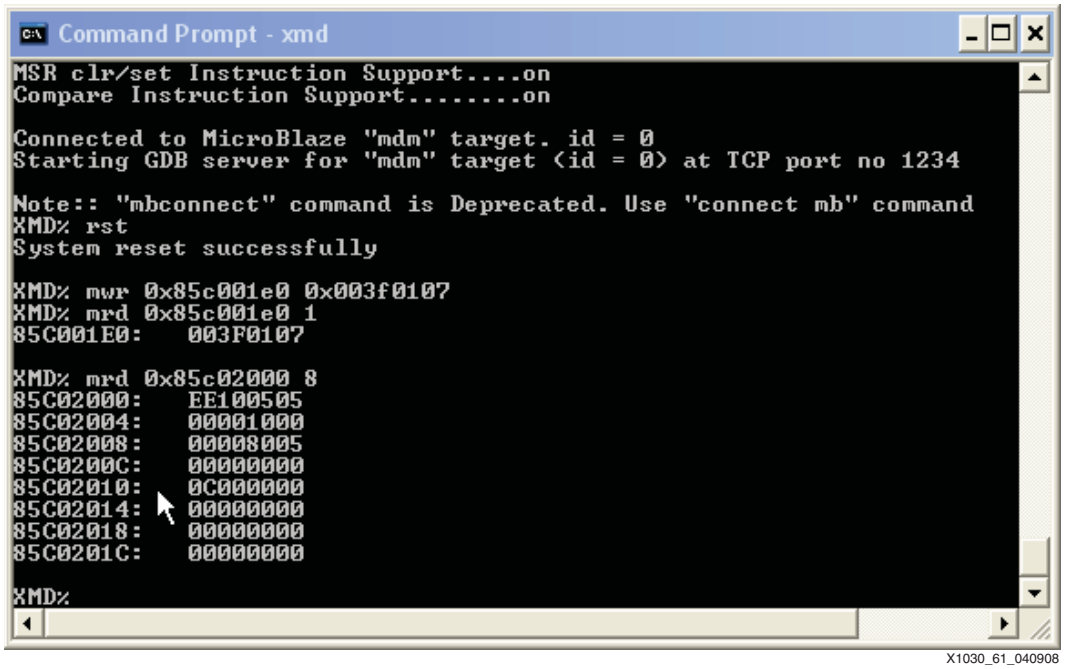


Figure 61: **XMD Read of PLBv46 Endpoint Bridge Registers**

After invoking PCItree and running a scan, Figure 62 shows the ML505 PLBv46 Endpoint Bridge detected as `Other; Memory Controller`, with Bus Number 3, Device Number 0, Function Number 0, or BDF = 3.0.0.The Xilinx Vendor ID and Device ID are displayed. In its Configuration Space Header, BAR0 has a value of `0x0000000C`. The address, 0xE000000 in this example, varies for different BIOS setups.

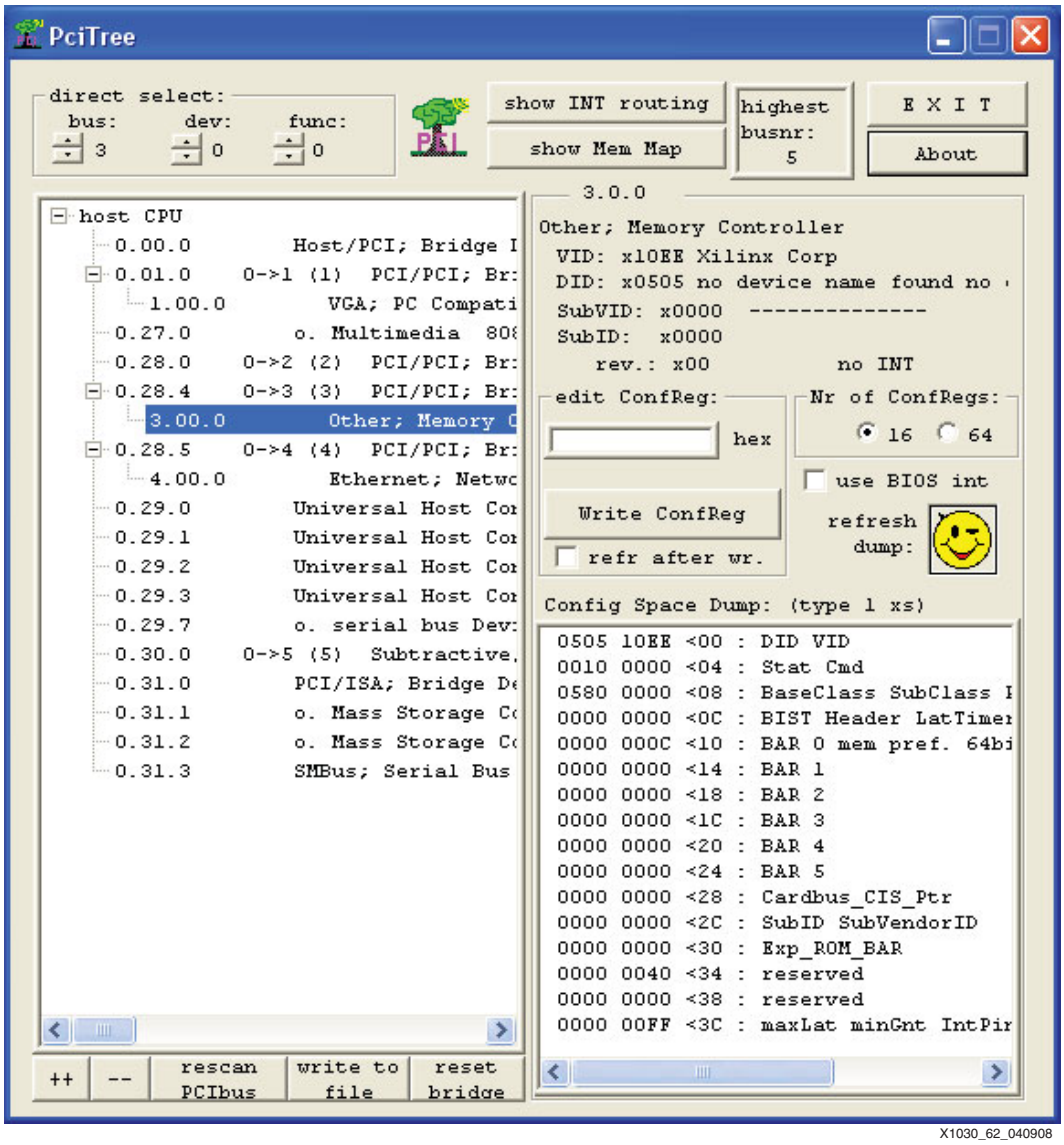


Figure 62: **PCItree Scan**

To edit the registers in the Configuration Space Header (CSH), highlight the register in the CSH to edit and provide a value in the Edit Config Reg dialog box. As an example, select the Command Status Register, write `xFFFFFFFF` in the Edit Config Reg dialog box, click WriteConfReg, and then click **Refresh Dump** to see the new value of the Command Status Register (CSR) displayed. The CSR value is not 0xFFFFFFFF because some bits in the CSR are reserved.

Click on BAR0 and use the edit ConfReg dialog box to change the value of BAR0 to xE000000C as shown in Figure 63. Click Write ConfReg and then Refresh Dump. The new value of BAR0 is displayed.

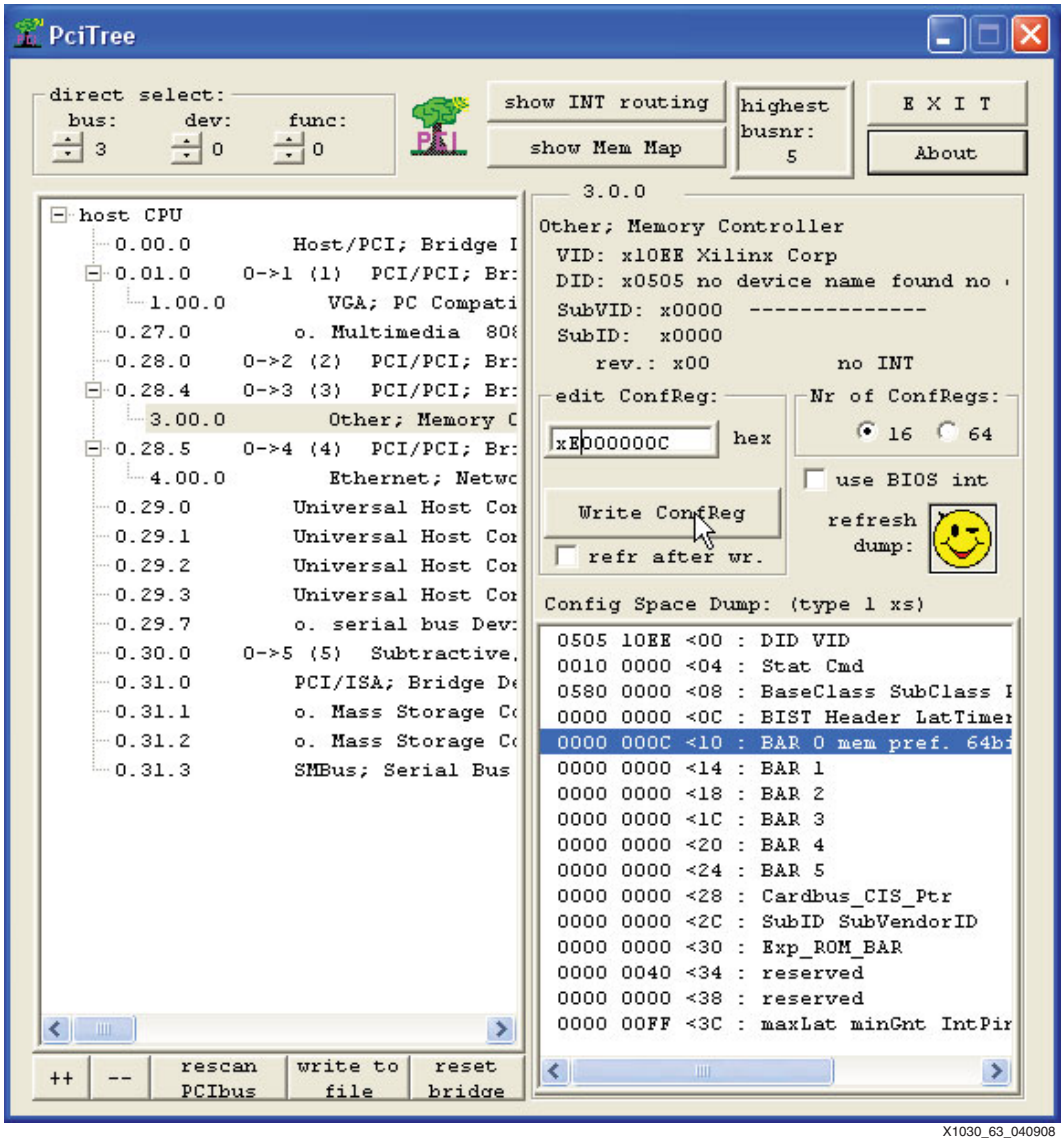


Figure 63: **Defining BAR0 in PCItree**

Figure 64 is XMD output which shows that BAR0 has been written as 0xE000000C. The XMD **mrd** command also shows that the data in the initial 8 addresses in XPS BRAM is 0x00000000.

As noted earlier, the XMD displays data in Big Endian formant while the x86 displays data in Little Endian format.

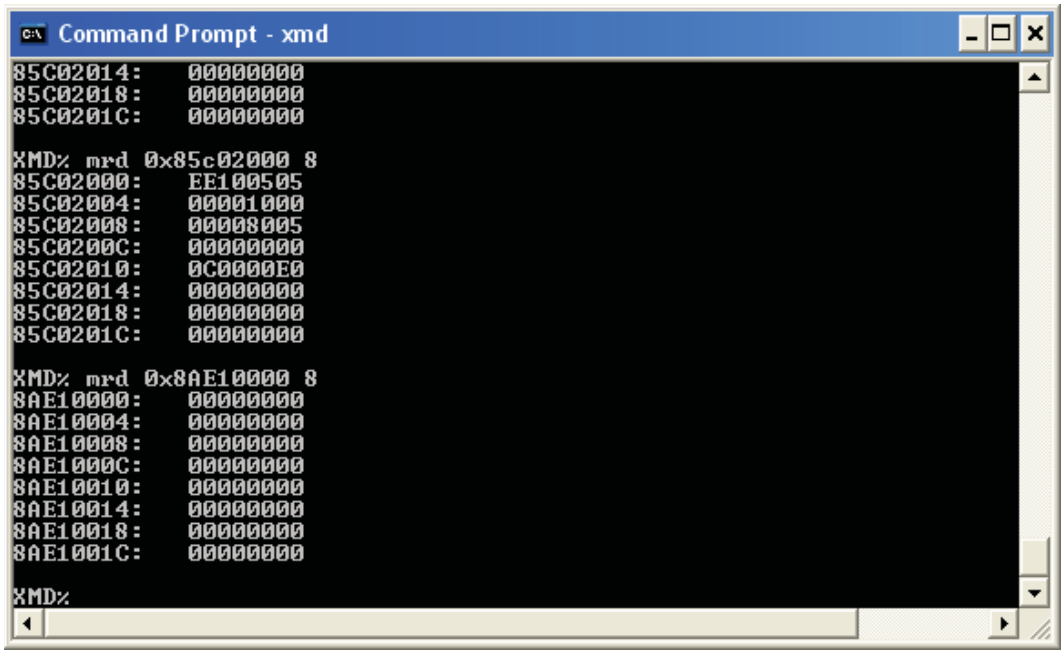
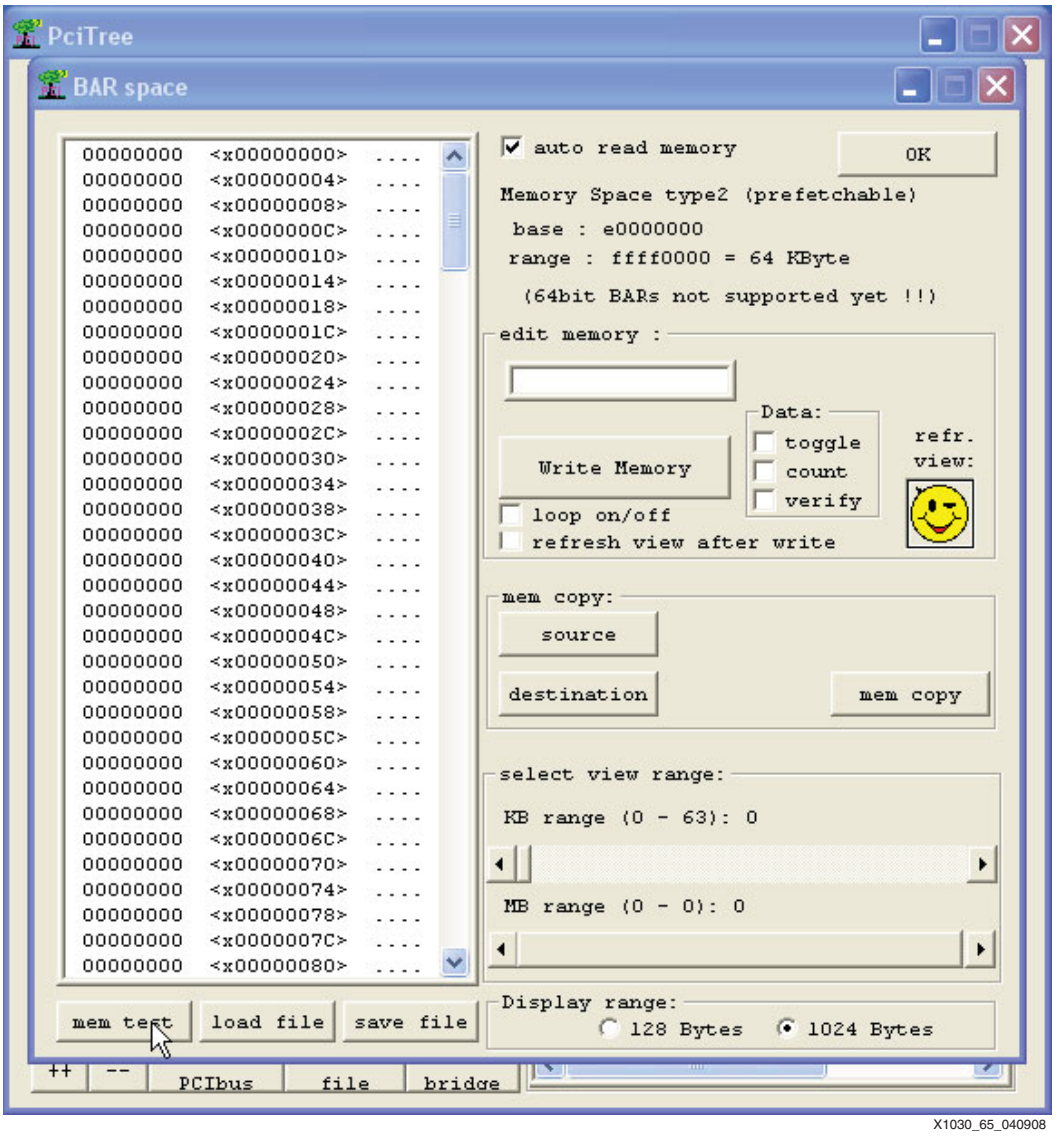


Figure 64: **XMD showing the Configuration Space Header, XPS BRAM**

Figure 65 shows the memory test for PCI tree. To run the memory test, click on **Mem Test** at the lower left of the BAR Space GUI. Check A**uto Read Memory** at the top of the BAR Space GUI to display memory values in the left side of the display. To edit a memory location, highlight the location to be edited, and enter the value in the **Edit memory** dialog box. Click on Write Memory. To view the results, click on the Refr. View icon.



X1030_65_040908

Figure 65: **Running PCItree Memory Test**

Figure 66 shows the results of running the memory test. The leftmost column shows the count pattern used for data. The count increments for even addresses and decrements on odd addresses. With the PCItree read of BAR0, the data is the count value specified in the PCItree memory test.The results (No Errors) are provided.

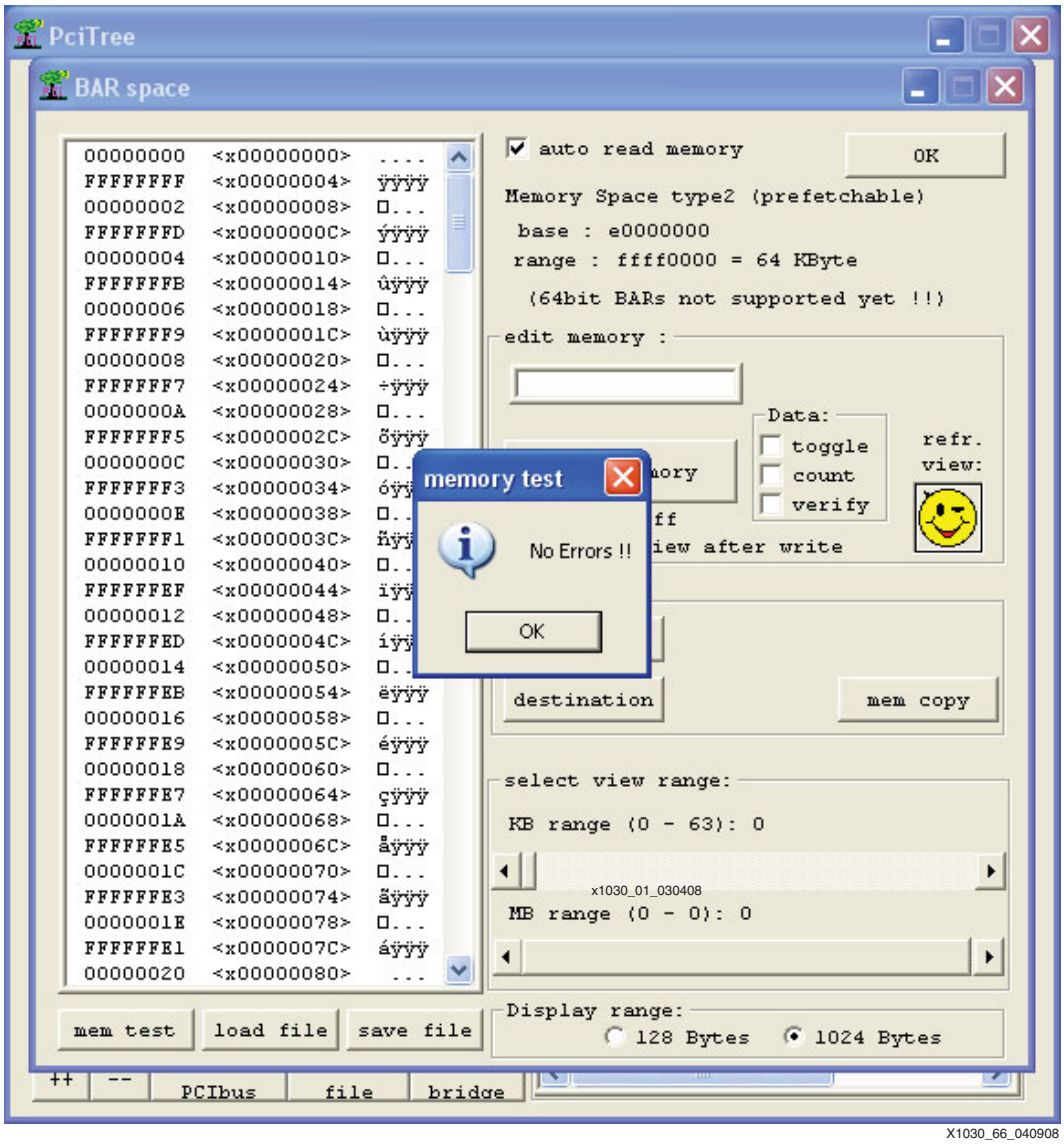


Figure 66: **PCItree Memory Test Results**

The ML505 memory written/read is the BRAM and/or DDR2 defined in the system.mhs and addressed with the PLBv46 Endpoint Bridge C_PCIBAR2IPIFBAR_* generics. In this reference system, two PLBv46 Endpoint Bridge BARs are active. The C_PCIBAR2IPIFBAR_0 generic points to the ML505 BRAM located at 0x8AE10000.

After writing the ML505 BRAM using PCI tree Edit Memory, XMD can be used to verify BRAM (or DDR2 if the BAR is enabled) from the PLBv46 side.

Figure 67 shows XMD verification that the XPS BRAM contains the data written by PCItree using XMD commands.

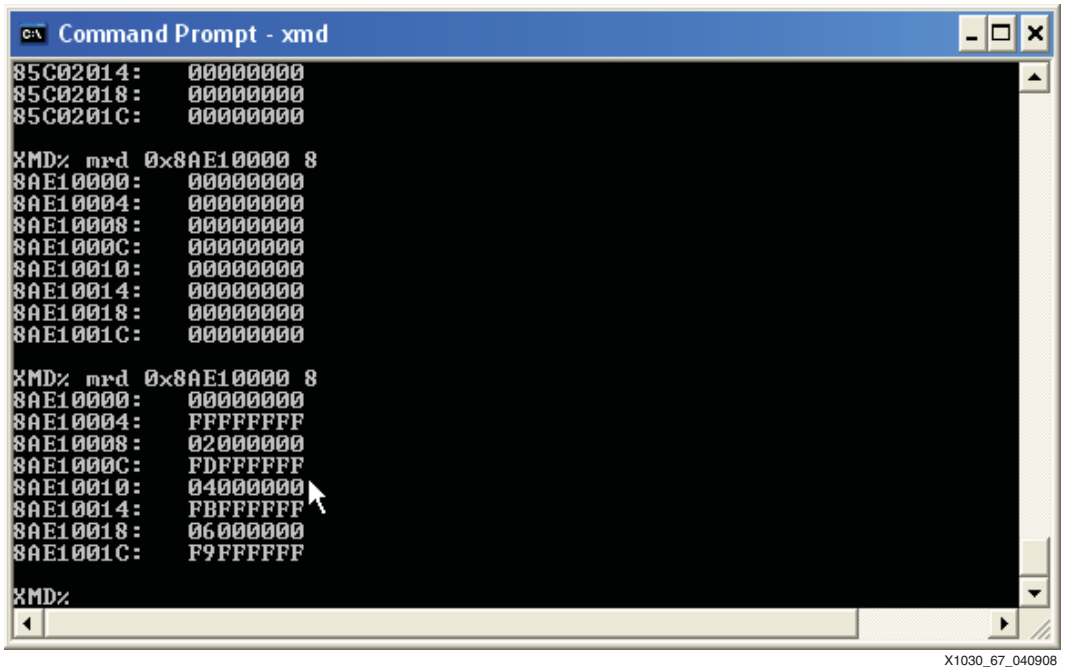


Figure 67: **XMD Verification of PCItree Write Operation**

In the next two figures, XMD is used to write XPS BRAM, which is then read by PCItree.

Figure 68 shows the writing and reading of 0x12345678 to the first four locations in XPS BRAM.

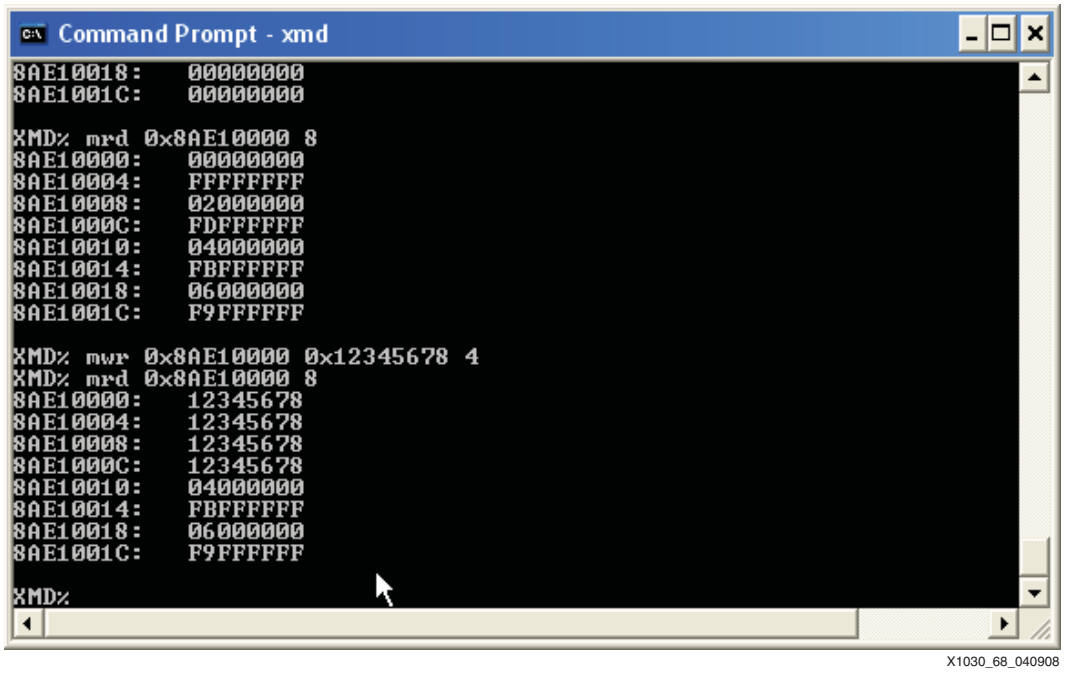


Figure 68: **Writing XPS BRAM using XMD**

Figure 69 shows a PCItree read of XPS BRAM. The first four locations are read as 0x1234567.

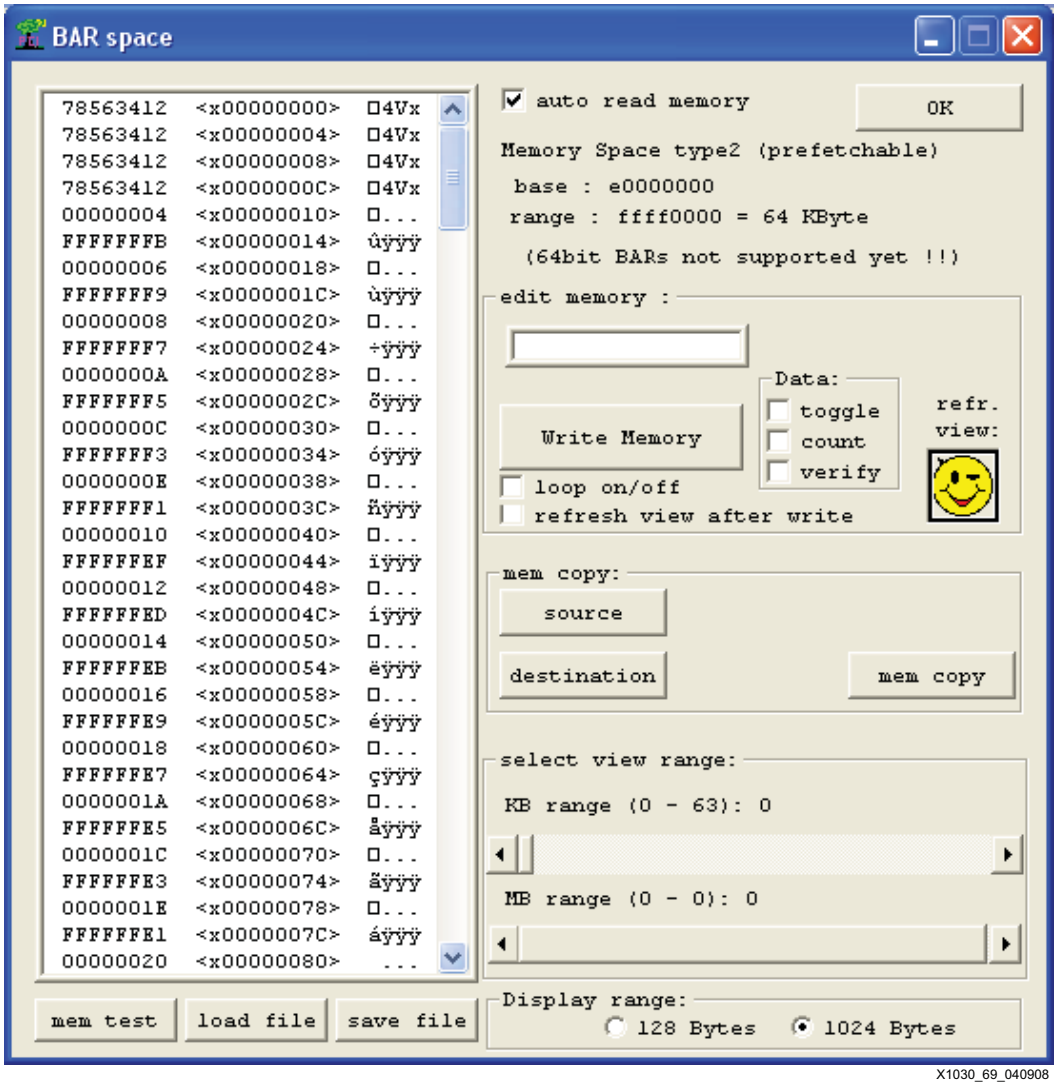


Figure 69: **PCItree Read of XPS BRAM**

Memory Endpoint Test

The Memory Endpoint Test (MET) is run on a PC with the ML505 inserted into a PCIe slot. MET provides a simple method of writing and reading memory. Like PCItree, the ML505 memory written/read is the BRAM and/or DDR2 defined in the system.mhs, and addressed with the PLBv46 Endpoint Bridge C_PCIBAR2IPIFBAR_* generics.

The MET requires the installation of the Xilinx Virtex-5 PCIe Endpoint Driver. The Xilinx application note XAPP1022 Using the Memory Endpoint Test (MET) Driver with the Programmed Input/Output (PIO) Example Design for PCI Express Endpoint Cores provides instructions on setting up and running the MET. XAPP1022 uses the PCIe Endpoint Block Plus core driven by the PIO interface. This section uses MET to write and read ML505 memory using the PLBv46 Endpoint Bridge.

Pages 6-11 of XAPP1022 provide instructions for installing the Xilinx Virtex-5 PCIe Endpoint Driver.

Figure 70 shows the invocation of the Memory Endpoint Test. The values for the Device Number, Vendor Number and the address indicate that the PLBv46 Endpoint Bridge on the ML505 is detected.

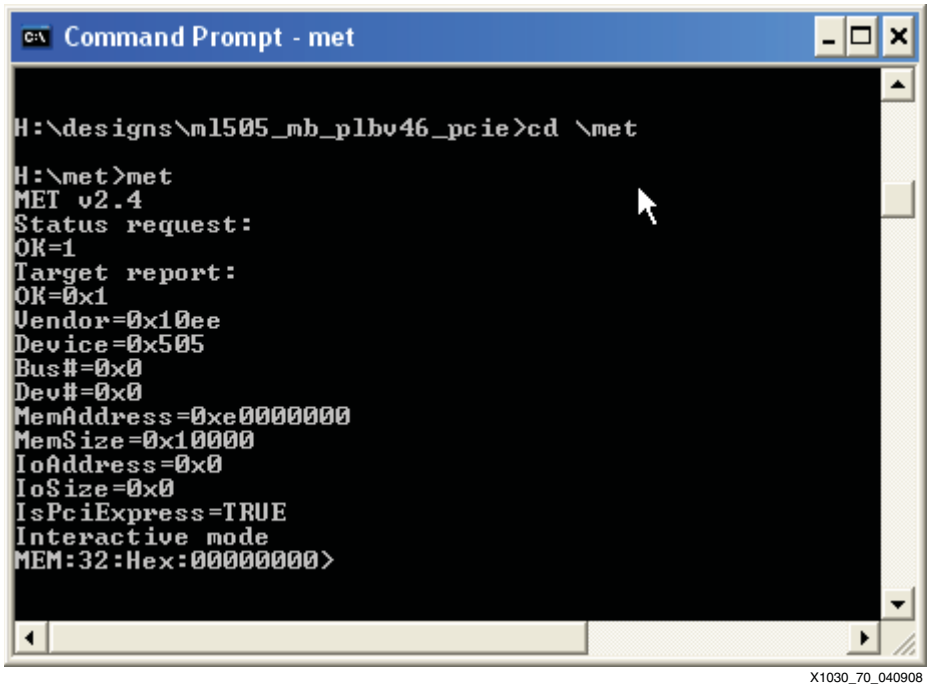


Figure 70: **Invoking the Memory Endpoint Test**

Pages 11-15 of XAPP1022 provide detailed instructions on using the MET to test transfers to PLBv46 Endpoint Bridge memory.

Figure 71 shows basic read and write operations using the MET. In the figure, the Display (d), Location (l), and Set (s) instructions illustrate basic memory read and write transactions.

The command

```
d 40
```

causes the values of 40 current memory locations to be displayed. The values displayed (`00000000 FFFFFFFF 00000002 FFFFFFFD ...`) are the same as the values displayed by PCItree in Figure 23 because this test was run shortly after the PCItree tests.

The location command

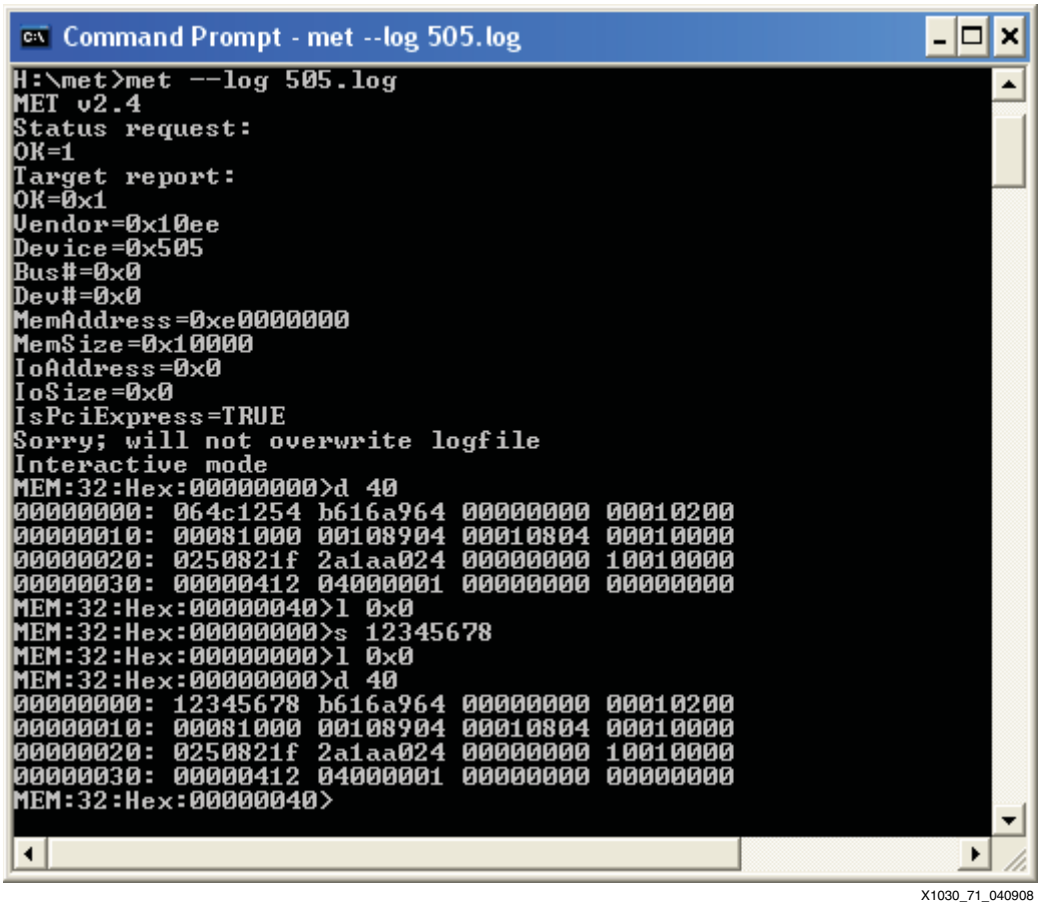
```
l 0
```

moves the address to location 0x00000000. All addresses are offset addresses from the BAR start address.

The set command

```
s 12345678
```

is a memory write to the current address. In the figure, after the write of 0x12345678, the address pointer is move back to location `0x00000000` (l 0), and the contents of the memory is re-displayed using d 40. The 0x12345678 value just written at location `0x00000000` is displayed.



X1030_71_040908

Figure 71: **Running the Memory Endpoint Test**

Using ChipScope with the PLBv46 Endpoint Bridge

ChipScope is used to debug hardware problems. Debugging is done at either the system or PLBv46 Endpoint Bridge level. To analyze PLBv46 Endpoint Bridge internal signals, insert the ChipScope cores into implementation/pcie_bridge_wrapper.ngc. To analyze signals involving multiple cores, insert the ChipScope cores into system.ngc. The flow for using the two debugging methods differs. Below, an outline of the steps for debugging at the system level is provided. This is followed by a detailed list of steps for debugging at the core level.

Inserting ChipScope at the System Level

The following steps insert the ChipScope cores into the system.

1. In XPS, select **Hardware** → **Generate Netlist.**
2. From the command prompt in the implementation directory, run

```
ngcbuild -i system.ngc system2.ngc
```

3. Copy `the chipscope/ml505_mb_plbv46_pcie.cdc` file to the project area (usually either one directory above the chipscope directory or the implementation directory).
4. Invoke ChipScope Inserter. To specify the input in the **Input Design Netlist** window, browse to the system2.ngc file created in step 2. Define the Clock, Trigger, and Data signals in Inserter, and generate the ICON and ILA cores.
5. From ML505_mb_plbv46_pcie/implementation, copy the file displayed in the Inserter Output Design Netlist window, usually implementation/system2.ngo, to implementation/system.ngc.
6. In XPS, run **Hardware** → **Generate Bitstream**.

Inserting ChipScope in the PLBv46 Endpoint Bridge

The `ml505_mb_plbv46_pcie/chipscope/plbv46_pcie.cdc` file is used to insert a ChipScope ILA core into the pcie_bridge_wrapper core. Do the following steps to insert a core and analyze PLBv46 Endpoint Bridge signals with ChipScope.

1. Invoke XPS. Run **Hardware** → **Generate Netlist**.

2. Copy `chipscope/plbv46_pcie.cdc` file to the project area (usually either one directory above the chipscope directory or the implementation directory).

3. Run **Start** → **Programs** → **ChipScope Pro** → **ChipScope Inserter**

4. From ChipScope Inserter, run **File Open** → **plbv46_pcie.cdc**.

Figure 72 shows the ChipScope Inserter setup GUI after **File Open** → **plbv46_pcie.cdc**.

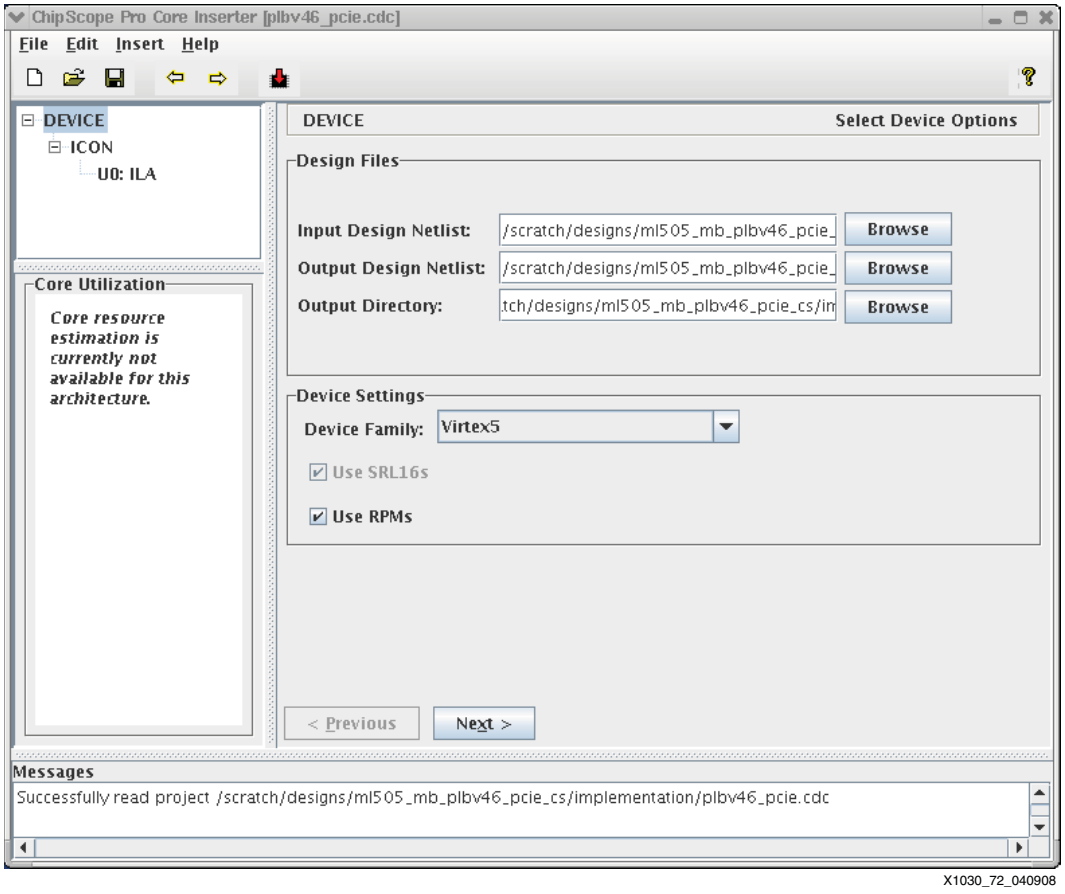


Figure 72: **Opening plbv46_pcie.cdc**

5. The plbv46_pcie.cdc provides a good starting point for analyzing designs. In most analyses, additional nets are needed. Figure 73 shows the GUI for making net connections. Click **Next** four times to move to the Modify Connections window. Select Modify Connections. The Filter Pattern is used to find net(s). As an example of using the Filter Pattern, enter `*ack*` in the dialog box to locate acknowledge signals such as SI_AddrAck. In the Net Selections area, select either Clock, Trigger, or Data Signals. Select the net and click **Make Connections**.

Correct Clock, Trigger, and/or Data signals displayed in red.

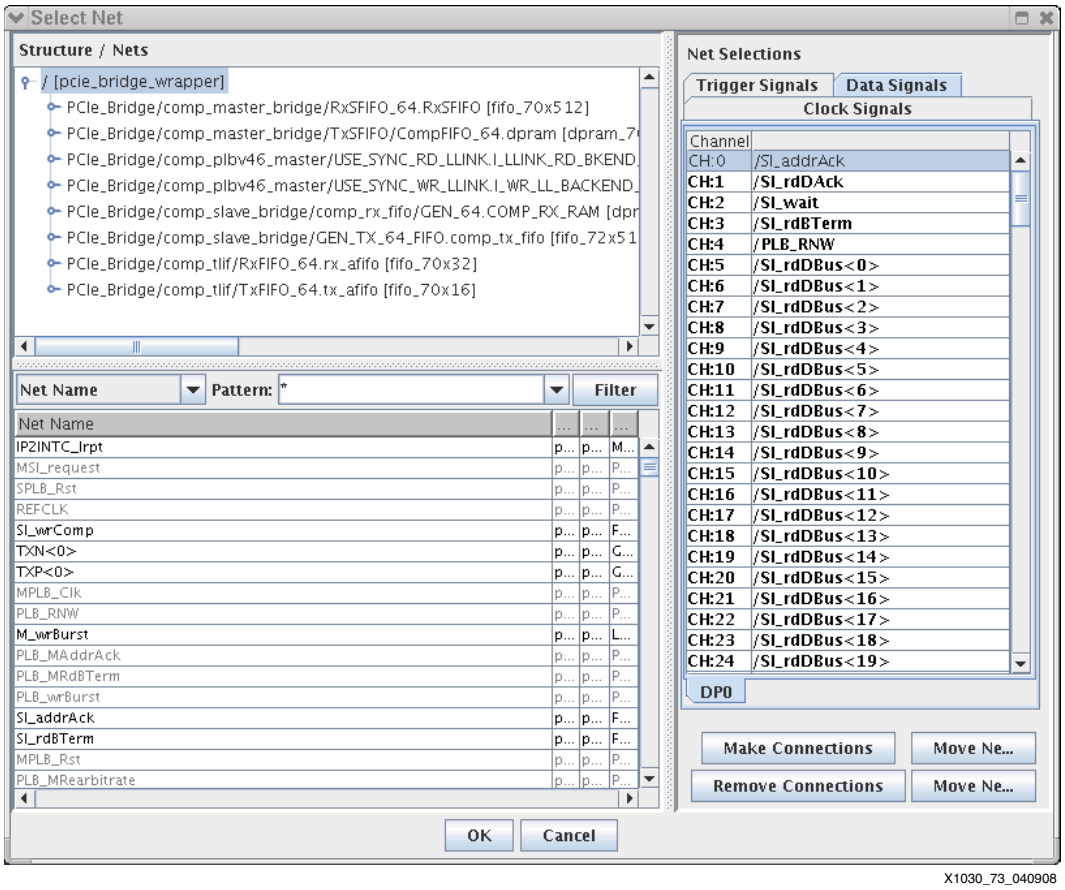


Figure 73: **Inserter Data Signals**

6. Click **Insert** to insert the core into pcie_bridge_wrapper.ngo. In the `ml505_mb_plbv46_pcie/implementation` directory, copy `pcie_bridge_wrapper.ngo` to `pcie_bridge_wrapper.ngc`.

8. In XPS, run **Hardware** → **Generate Bitstream** and **Device Configuration** → **Download Bitstream**. Do not rerun **Hardware** → **Generate Netlist**, as this overwrites the `implementation/pcie_bridge_wrapper.ngc` produced by the step above. Verify that the file size of the pcie_bridge_wrapper.ngc with the inserted core is significantly larger than the original version.

9. Invoke ChipScope Pro Analyzer by selecting

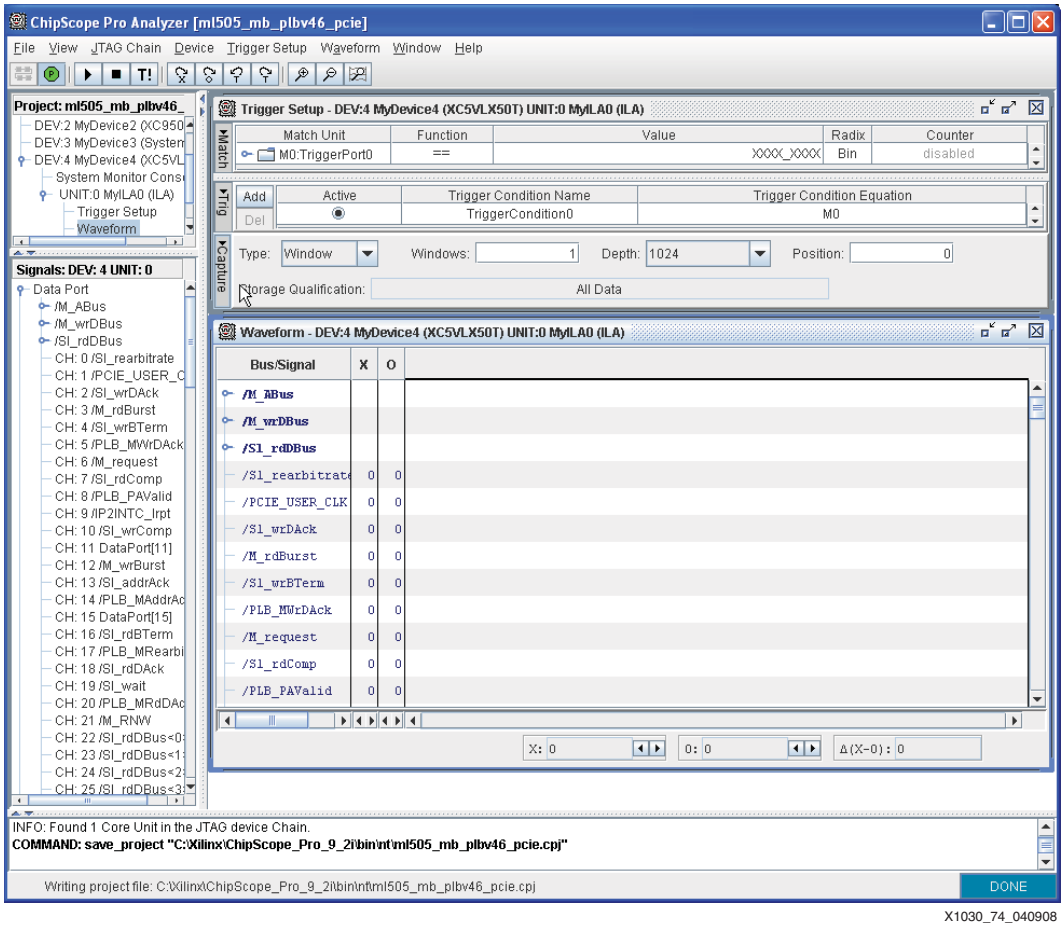
Start → **Programs** → **ChipScope Pro** → **ChipScope Pro Analyzer**

Click on the Chain icon located at the top left of Analyzer's GUI. Verify that the message in the transcript window indicates that an ICON is found.

10. The ChipScope Analyzer waveform viewer displays signals named `DATA*`. To replace the `DATA*` signal names with the familiar signal names specified in ChipScope Inserter, select **File** → **Import** and browse to plbv46_pcie.cdc in the dialog box.

The Analyzer waveform viewer is more readable when buses rather than discrete signals are displayed. Select the **SI_rdDBus<*>** signals, click the right mouse button, and select **Add to Bus** → **New Bus**. With **SI_rdDBus** in the waveform viewer, select and delete the discrete **SI_rdDBus<*>** signals. The signals are displayed as buses in Figure 74.

Note: The Reverse Bus Order operation is useful for analyzing buses in Analyzer.



X1030_74_040908

Figure 74: **ChipScope Pro Analyzer Waveform**

11. Set the trigger in the Trigger Setup window as shown in Figure 75. The trigger used depends on the problem being debugged. Simple triggers are PA_Valid, SI_AddrAck, SI_wrComp.

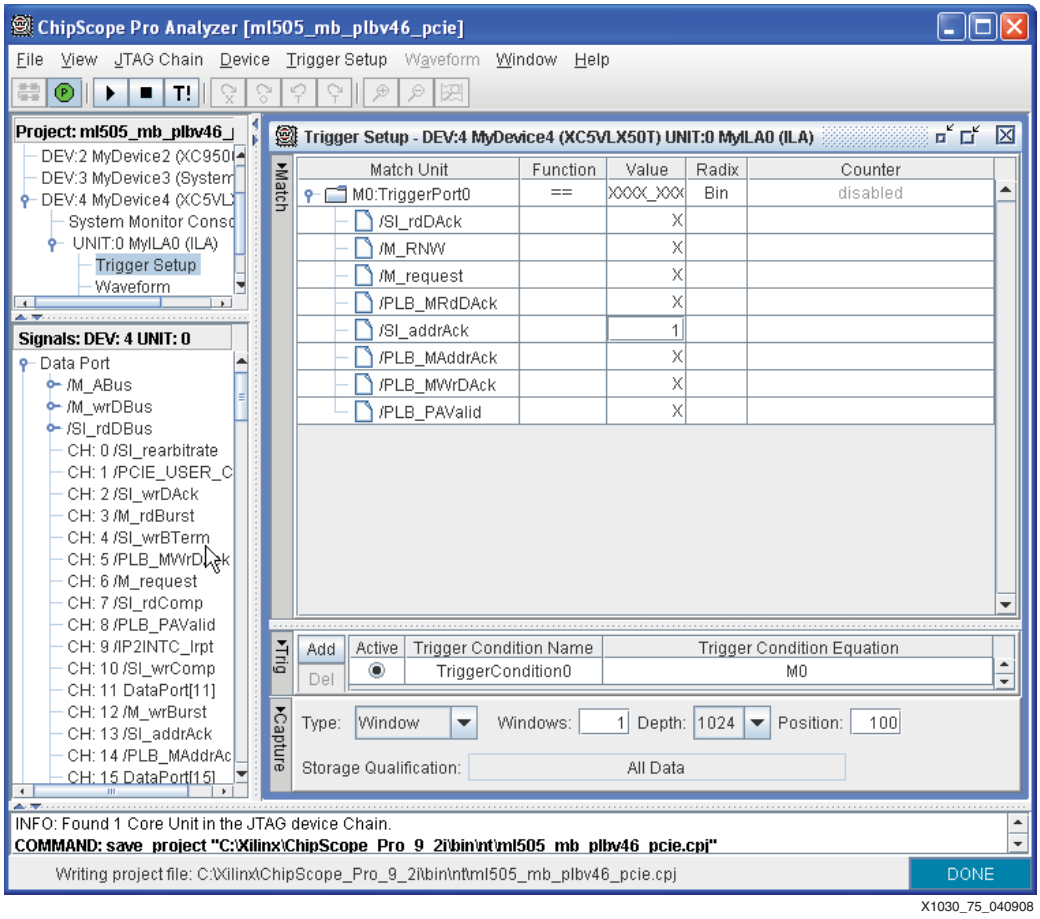


Figure 75: **ChipScope Analyzer Trigger Setup**

12. Arm the trigger by selecting **Trigger Setup** → **Arm**, or clicking on the **Arm** icon as shown in Figure 76.

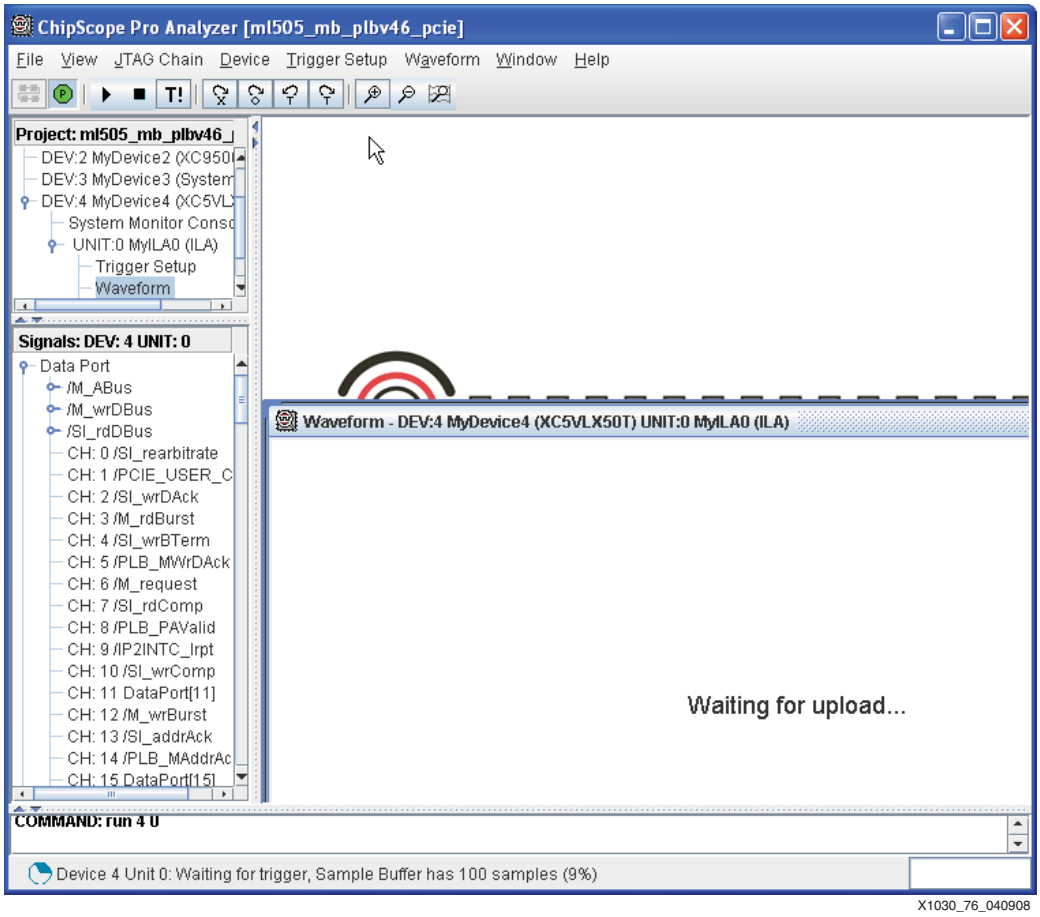


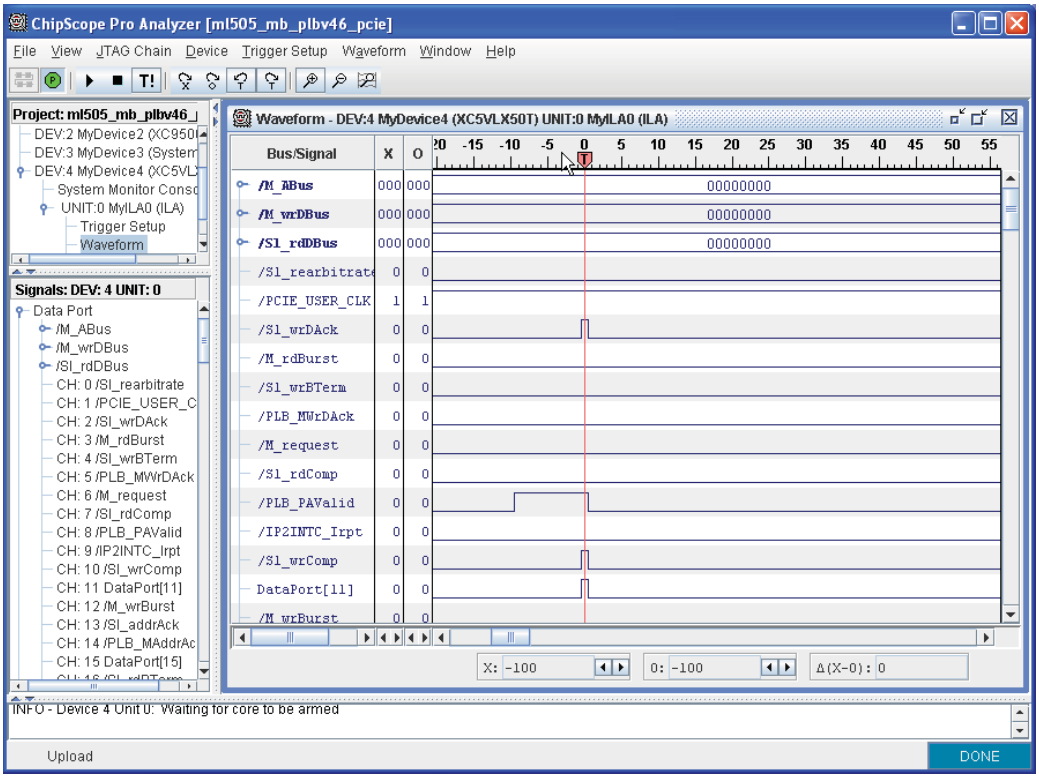
Figure 76: **ChipScope with Trigger Armed**

13. Run **XMD** or **GDB** to trigger patterns which cause ChipScope to display waveform output. For example, set the trigger to SI_addrAck, arm the trigger, and run

```
xmd -tcl xmd_commands/dma.tcl
```

at the command prompt. This produces signal activity in the Analyzer waveform viewer.

14. ChipScope results are analyzed in the waveform window, as shown in Figure 77. This figure shows the bus signals generated in Step 10.



X1030_77_040908

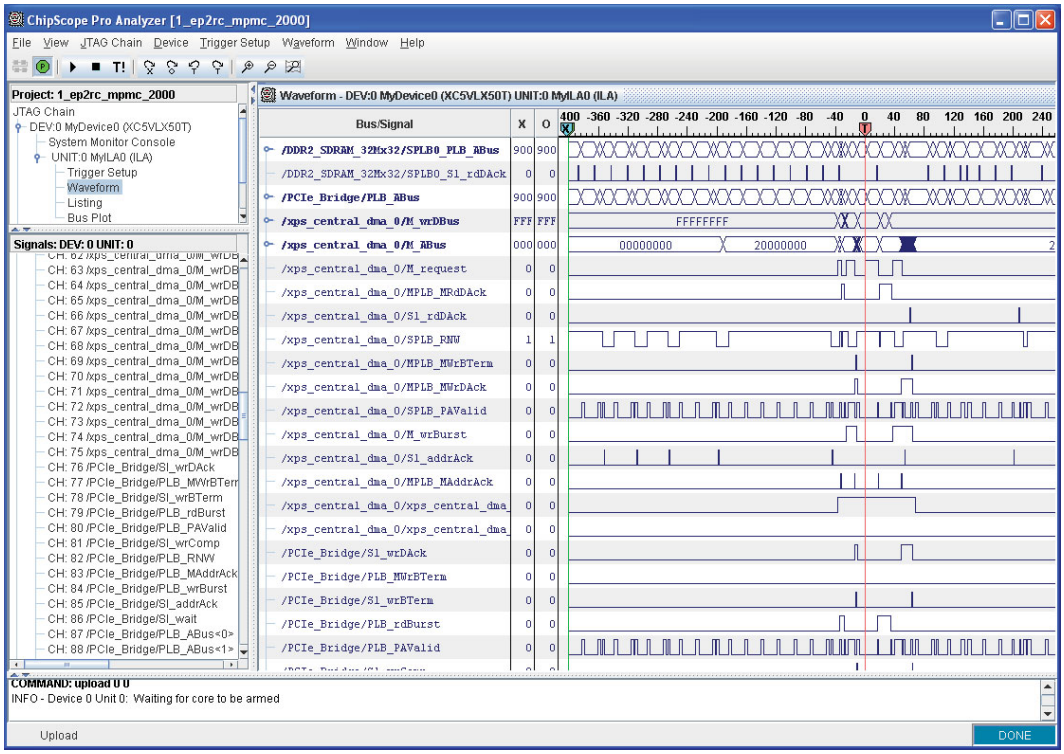
Figure 77: **ChipScope Pro Analyzer Triggered**

To share the results with remote colleagues, save the results in the waveform window as a Value Change Dump (vcd) file. The vcd files can be translated and viewed in most simulators. The `vcd2wlf` translator in ModelSim reads a vcd file and generates a waveform log file (wlf) file for viewing in the ModelSim waveform viewer. Select **File** → **Open Database** to open the vcd file in the Cadence Design System, Inc. Simvision design tool.

After running ChipScope, it is sometimes necessary to revise the Trigger or Data nets, or both, used in a debug operation. Saving Inserter and Analyzer projects simplifies this procedure. The saved project can be re-opened in Inserter, and edits can be made.

Figure 78 is the waveform output of a ChipScope inserted into the reference system when running the endpoint to root complex performance tests.

Memory, XPS Central DMA and PLBv46 Endpoint Bridge transactions are monitored simultaneously. The trigger is PCIe_bridge/comp_slave_bridge/sig_request_complete. The ml505_mb_plbv46_pcie_scs.cdc is included in the chipscope directory.



X1030_78_040908

Figure 78: **System Debugging Using ChipScope**

Reference Design Matrix

The reference design matrix is shown in Table 4.

Table 4: **Reference Design Matrix**

General	
Developer Name	Xilinx
Target devices (stepping level, ES, production, speed grades)	Virtex-5 XC5VLX50T (Production Silicon)
Source code provided	No
Source code format	VHDL
Design uses code/IP from an existing reference design/application note, 3rd party, or CORE Generator software	No
Simulation	
Functional simulation performed	No
Timing simulation performed	No
Testbench used for functional simulations provided	No
Testbench format	N/A
Simulator software used/version (i.e., ISE software, Mentor, Cadence, other)	N/A
SPICE/IBIS simulations	No
Implementation	
Synthesis software	XST
EDK Software	EDK10.1i
Implementation software tools used/versions	ISE10.1i
Static timing analysis performed	Yes
Hardware Verification	
Hardware verified	Yes
Hardware platform used for verification	ML505

References

1. UG197 Virtex-5 Integrated Endpoint Block for PCI Express Designs User Guide
2. UG201 Virtex-5 FPGA ML555 Development Kit for PCI and PCI Express Designs User Guide (v1.4) March 10, 2008
3. XAPP1022 Using the Memory Endpoint Driver (MET) with the Programmed Input/Output Example Design for PCI Express Endpoint Cores
4. LeCroy PCI Express Multi-Lane Exerciser User Manual Version 5.0
5. SpekChek User Manual Version 6.5
6. Catalyst - PCI Express Bus Protocol Analyzer/Exerciser User's Guide

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
04/25/08	1.0	Initial release.
5/6/08	1.0.1	Made minor non-technical edits.

Notice of Disclaimer

Xilinx is disclosing this Application Note to you "AS-IS" with no warranty of any kind. This Application Note is one possible implementation of this feature, application, or standard, and is subject to change without further notice from Xilinx. You are responsible for obtaining any rights you may require in connection with your use or implementation of this Application Note. XILINX MAKES NO REPRESENTATIONS OR WARRANTIES, WHETHER EXPRESS OR IMPLIED, STATUTORY OR OTHERWISE, INCLUDING, WITHOUT LIMITATION, IMPLIED WARRANTIES OF MERCHANTABILITY, NONINFRINGEMENT, OR FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT WILL XILINX BE LIABLE FOR ANY LOSS OF DATA, LOST PROFITS, OR FOR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, OR INDIRECT DAMAGES ARISING FROM YOUR USE OF THIS APPLICATION NOTE.